VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

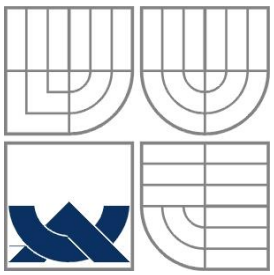
INTEGRACE HLASOVÝCH TECHNOLOGIÍ NA MOBILNÍ PLATFORMY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

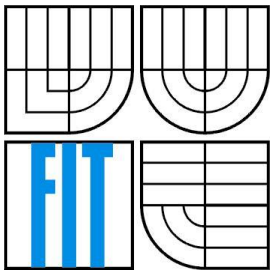
AUTOR PRÁCE
AUTHOR

Bc. SERGIJ ČERNIČKO

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTEGRACE HLASOVÝCH TECHNOLOGIÍ NA MOBILNÍ PLATFORMY

INTEGRATION OF VOICE TECHNOLOGIES ON MOBILE PLATFORMS

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. SERGIJ ČERNIČKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR SCHWARZ, Ph.D.

BRNO 2013

Abstrakt

Cílem práce je seznámit se s metodami a technikami využívanými při zpracování řeči. Popsat současný stav výzkumu a vývoje řečových technologií. Navrhnout a implementovat serverový rozpoznávač řeči, který využívá BSAPI. Integrovat klienta, který bude využívat server pro rozpoznání řeči, do mobilních slovníků společnosti Lingea.

Abstract

The goal of the thesis is being familiar with methods a techniques used in speech processing. Describe the current state of research and development of speech technology. Project and implement server speech recognizer that uses BSAPI. Integrate client that will use server for speech recognition to mobile dictionaries of Lingea company.

Klíčová slova

Hlas, řeč, rozpoznávání řeči, automatické rozpoznávání řeči, ASR, dynamické borcení času, DTW, skryté Markovy modely, HMM, Android, slovník

Keywords

Voice, speech, speech recognition, automatic speech recognition, ASR, dynamic time wraping, DTW, hiden Markov models, HMM, Android, dictionary

Citace

Černičko Sergij: Integrace hlasových technologií na mobilní platformy, diplomová práce, Brno, FIT VUT v Brně, 2013

Integrace hlasových technologií na mobilní platformy

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Schwarce, Ph.D.
Další informace mi poskytl Mgr. Petr Lokaj ze společnosti Lingea s.r.o.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Sergij Černičko
25.5.2013

Poděkování

Rád bych poděkoval vedoucímu práce Petru Schwarzovi za čas a rady, které mi poskytl. Děkuji také Petrovi Lokajovi za jeho informace, které mi pomohli při implementaci.

Obsah

1 Úvod

Komunikace prostřednictvím mluvené řeči je základní a nejpřirozenější způsob přenosu informace mezi lidmi. Při současném vývoji hardwaru a zvyšujících se možnostech výpočetní techniky se vědci a technici snaží o to, aby se počítač či mobilní zařízení mohlo stát plnohodnotným partnerem člověka v mluveném dialogu. Takový způsob komunikace může být pro člověka velice prospěšný a často mu usnadní práci. Nejde pouze o usnadnění práce, ale tento způsob může pomoci lidem, trpícím nějakým postižením, snadněji manipulovat s technikou. Při zpracování řeči musíme algoritmicky a technicky vyřešit několik relativně komplikovaných úloh, které se týkají zpracování řečového signálu a automatického rozpoznávání řeči.

Plnohodnotný rozpoznávač plynulé řeči bez jakýchkoli omezení není v současné době zatím ještě dostupný, ale je ve fázi výzkumu, na kterém se podílejí vědci z celého světa. Je to způsobeno především stálými obtížemi s rozpoznáváním zejména spontánní řeči. Oproti minulosti se zdokonalováním automatického rozpoznávání řeči začal růst počet slov obsažených v rozpoznávacím slovníku, z několika stovek v osmdesátých letech na několik set tisíc slov dnes. Také kvalita rozpoznávání řeči přešla z čistých laboratorních dat ke spontánním hovorům v rušném prostředí. Dnešní systémy už dokáží pracovat v konkrétním prostředí až s 97% úspěšností.

I přes to, že systémy hlasové komunikace mají zatím celou řadu nedořešených problémů, tak dochází stále častěji k jejich praktickému nasazování v průmyslové a společenské praxi. Tyto systémy bývají nasazovány, jde-li o hlasovou komunikaci s databázovými systémy. Toto se dá využít při vzdáleném přístupu k informačním a rezervačním systémům a objednávání zboží po telefonu. Výhodou je to, že tyto služby mohou být zákazníkovi k dispozici nonstop a z jakéhokoliv místa. Všeobecné využití naleznou rozpoznávače při ovládání strojů a zařízení hlasovými příkazy nebo automatickém přepisu mluveného diktátu.

U mobilních zařízení se začali objevovat inteligentní asistenti, kteří reagují na povely od mluvčího a dle nich zahájí nějakou činnost. Dokáží s uživatelem vést rozhovor a odpovídají mu na otázky (např. Kolik je hodin?, Jaké je počasí?, atd.), nastavují či ovládají zařízení, vyhledávají na webu, atd. Pracuje se již i na velmi perspektivních aplikacích automatického překladu z řeči do řeči. Tyto systémy předpokládají vstup v jednom jazyce. Provedou automatický překlad do druhého jazyka, korekci (gramatika) a nakonec ještě syntézu.

V mé práci je automatické rozpoznávání řeči implementováno jako serverová aplikace, čímž dostanu velký výpočetní výkon pro převod řeči na text. Klientská část je zakomponována do mobilních slovníků společnosti Lingea (konkrétně aplikace HandyLex) a posílá na server data získaná z nahrávání. Server provede rozpoznání a vrátí pouze pole s několika nejlepšími výsledky. V aplikaci potom provedu překlad těchto slov a zobrazím uživateli výsledky. Celá komunikace by měla být po částech a ne pouze poslat data najednou, tím dostanu tzv. real-time zpracování, které urychlí běch aplikace (nemusím čekat na jednotlivé kroky a provádím více operací paralelně).

2 Zpracování řeči

Zvuk můžeme definovat obecně jako mechanické kmitání, které je charakterizováno parametry pohybu částic pružného prostředí nebo u vlnového pohybu parametry zvukového pole. Část zvuků se projevuje jako slyšitelný zvuk, který má frekvenci od 16 Hz do 20 kHz (silně individuální, jen málokdo je schopen vnímat celé pásmo). Podle frekvence je možno zvuk rozdělit ještě na infrazvuk (do 16 Hz) a ultrazvuk (nad 20 kHz). Řeč se tedy přenáší komunikačním kanálem ve formě akustických vln (akustický signál). Komunikačním kanálem rozumíme prostředí, kterým je akustický signál přenášen od svého zdroje až po příjemce. V akustickém signálu je zakódováno několik druhů informace.

2.1 Akustická a auditivní fonetika

Akustická a auditivní fonetika zaměřuje svou pozornost na výsledný signál lidské řeči bez ohledu na artikulační mechanismy, jimiž byl vytvořen.

Auditivní pohled [2] na lidskou řeč je mnohem starší, je založen na analýze a hodnocení řeči sluchem. Není ani zdaleka tak přesný jako pohled akustický, protože lidské ucho má omezenou možnost vnímání, a není ani možno jej dokonale objektivizovat, protože je do jisté míry poznamenán subjektem člověka, který výzkum provádí.

Pohled **akustické fonetiky** [2] je soustředěn na studium autentického, nezobecněného signálu lidské řeči, který je objektivními fyzikálními metodami a pomocí experimentu zkoumán a popisován v celé složitosti. Toto odvětví výzkumu se velmi rozvíjí v posledních desetiletích dík rozvoji fyzikálních metod analýzy zvuku a dík nebývalým možnostem vyhodnocovat zjištěné údaje pomocí počítače a verifikovat výsledky analýz pomocí modelů. Výsledky jsou pak využívány nejen pro hlubší poznání lidské řeči, ale mají také praktické využití v nejrůznějších oblastech komunikace mezi člověkem a strojem (hlasové vstupy, hlasové výstupy, apod.). Akustická analýza umožní také identifikaci mluvčího a má pak využití třeba v kriminalistice. Možnosti akustických zkoumání a jejich uvádění do praxe nejsou ani zdaleka vyčerpány.

Pro snímání řeči se používá mikrofon, který převádí zvuk na elektrický signál. Ten se digitalizuje a reprezentuje jako posloupnost řečových vzorků. Akustickými rysy řeči jsou intenzita zvuku, dále jeho výška, hlasitost, barva a rozdělení spektrální energie. V časové oblasti se řečový signál reprezentuje pomocí časového průběhu vlny, který zobrazuje vývoj amplitudy signálu v čase.

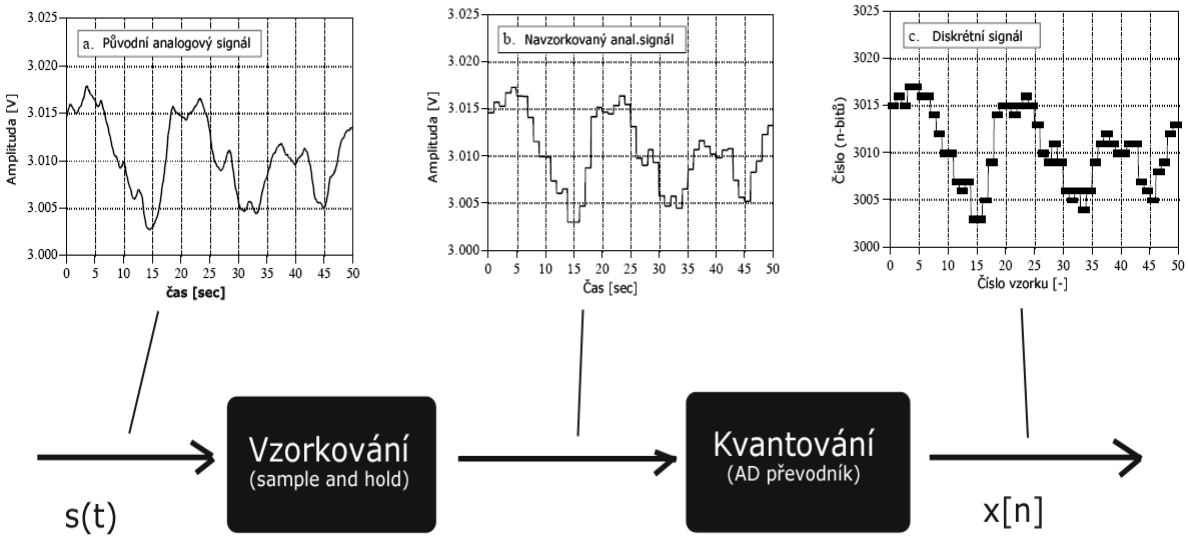
Frekvenční oblast je vhodná k popisu spektrálních vlastností řeči. K výpočtu spektrálních charakteristik se používá Fourierova transformace. Spektrální povaha signálu se zobrazuje pomocí tzv. amplitudového spektra, které znázorňuje vývoj amplitudy signálu ve frekvenci. Znělé úseky řeči v něm vystupují jako úzké spektrální vrcholky, střední frekvence těchto vrcholků jsou přitom v harmonickém vztahu s frekvenci základního tonu. Na druhou stranu spektra neznělých zvuků jsou v podstatě nahodilé. Celkový tvar spekter obou typů zvuků, tzv. spektrální obálka, pak vykazuje široké vrcholky a údolí, které odpovídají jednotlivým formantům a antiformantům hlasového traktu. Stav jejich středních frekvencí a šířek pásem v čase udává zabarvení hlasu odpovídajících zvuků (témbr). U znělých zvuků je většina akustické energie soustředěna na nižších frekvencích. Naproti tomu u neznělých zvuků převládají vysokofrekvenční složky.

3 Předzpracování zvukového signálu

Tato kapitola popisuje základy předzpracování zvukového signálu, které se používají před jeho vlastním zpracováním pomocí dalších metod.

3.1 Digitalizace

Pomocí digitalizace převedeme analogový spojitý signál na digitální. Digitalizace se provádí vzorkováním a kvantování signálu (obrázek 1). Ještě před samotnou digitalizací se používají **analogové filtry** [1] **(antialiasingové filtry)**, které jsou typu dolní propust, tedy spodní frekvence nechají a horní utlumí.



Obrázek 1: Příklad vzorkování a kvantování signálu [4]

3.1.1 Vzorkování

Vzorkovaný signál dostaneme tak, že původní signál vynásobíme něčím, co je periodické v čase (Diracův impuls). Jedná se tedy o rozložení definičního oboru na konečný počet podmnožin. Spektrum vzorkovaného signálu se periodizuje. Aby se nepřekrývali jednotlivé kopie původního spektra musí platit **Shanonův–Kotelnikovův–Nyquistův–vzorkovací teorém** [1]:

$$F_s > 2f_{max,} \tag{1}$$

neboli musím vzorkovat dvakrát rychleji, než je nejrychlejší frekvence v signálu.

3.1.2 Kvantování

Kvantování [3] je diskretizace oboru hodnot signálu. Vždy je to proces ztrátový a nevratný. Chyba kvantování má (skoro) vždy charakter náhodného šumu s rovnoměrným rozdělením. Někdy se před kvantováním přidá k původnímu signálu malé množství náhodného šumu (dithering), které by mělo chybu „vymaskovat“.

3.2 Segmentace na rámce

Rámce [3] jsou úseky signálu, na které je nutné řečový signál před dalším zpracováním rozdělit. Řečový signál pro metody odhadu parametrů by měl být stacionární. Ale není, proto jej dělíme na kratší úseky. Tam by měl být stacionární. Parametry rámců [3]:

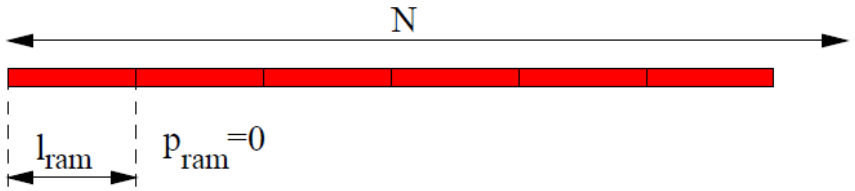
- **délka** (l_{ram}) – by měla být dostatečně malá, aby bylo možné pokládat signál na daném úseku za stacionární, ale na druhé straně dostatečně velká, aby bylo možné přesněji odhadnout požadované parametry. Kompromisem je délka respektující setrvačnost hlasového ústrojí, typicky 20-25 ms.
- **překrytí** (p_{ram}) by bylo vhodné:
    - **malé nebo žádné** – rychlý časový posun v signálu, malé nároky na paměť/procesor, hodnoty parametrů se však od jednoho rámce ke druhému mohou hodně měnit.
    - **velké** – zajišťuje pomalý časový posuv, „vyhlazené“ průběhy parametrů, avšak má velké nároky na paměť/procesor. Výsledné parametry mohou být navíc rámec od rámce příliš podobné, což jde proti požadavku statistické nezávislosti při rozpoznávání pomocí skrytých Markovových modelů.
- **posun rámce** ($s_{ram} = l_{ram} - p_{ram}$)

U **rámců bez překrývání** [3], kde $p_{ram} = 0$ (běžné pro kódování) dostaneme jejich počet prostým dělením a zaokrouhlením dolů. Předpokládáme, že poslední rámec, na který již nemáme dost vzorků, zahazujeme.

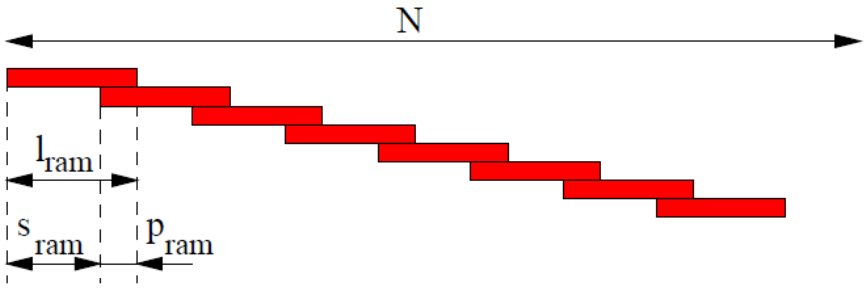
$$N_{ram} = \left\lfloor \frac{N}{l_{ram}} \right\rfloor \tag{2}$$

Pro **rámce s překrýváním** [3], kde $p_{ram} \neq 0$ (běžné pro rozpoznávání) je počet rámců (pokud je signál alespoň jeden rámec dlouhý) dán:

$$N_{ram} = 1 + \left\lfloor \frac{N - l_{ram}}{s_{ram}} \right\rfloor \tag{3}$$



Obrázek 2: Rámec bez překrytí [3]



Obrázek 3: Rámec s překrytím [3]

3.2.1 Výběr signálu do rámce

Zvukový signál v okolí rámce je periodický s periodou uvnitř rámce. Není-li perioda shodná s délkou rámce, popř. mají-li rámce nulové překrytí, můžeme se dopustit chyby ve zpracování. Proto při „vykrojení“ rámce je použito „okno“ (windowing function). Účelem použití okna je vybrat vzorky signálu v analyzovaném rámci a vyhladit jejich průběh přidělením váhy. Existuje několik typů, ale zde uvádím pouze dvě nejpoužívanější [3]:

- **pravoúhlé (rectangular) okno** – se signálem neudělá nic.
- **Hammingovo okno** – je nejpoužívanější, utlumí signál na okrajích.

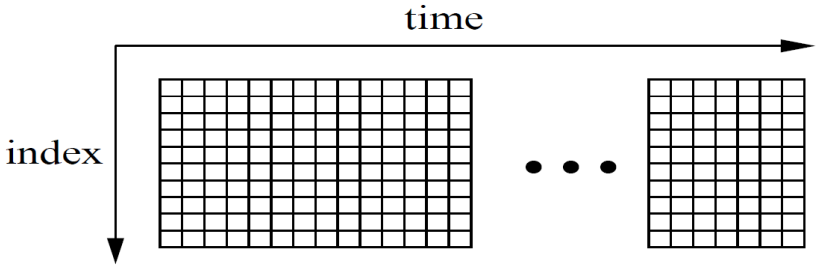
3.3 Parametrizace

Parametrizace [3] (anglicky „feature extraction“) vyjadřuje řečový signál omezeným množstvím hodnot. V klasické literatuře jsou metody popisu děleny [3]:

- **neparametrický popis** – je založen pouze na poznatcích o zpracování signálu (banky filtrů, Fourierova transformace, atd.)
- **parametrický popis** – je založen na poznatcích o tvorbě řeči

Druhá technika ovšem používá mnoho technik neparametrického popisu, takže tyto dvě skupiny není snadné (a někdy ani žádoucí) oddělit. Vypočtené hodnoty stejně vždy nazýváme parametry. Podle charakteru dělíme parametry [3]:

- **skalární –** jedno číslo na řečový úsek (krátkodobá energie nebo počet průchodů nulou)
- **vektorové** – sada čísel (vektor) na řečový úsek. Pokud je více řečových úseků, řadíme vektorové hodnoty do matic tak, že čas běží vodorovně (obrázek 4)



Obrázek 4: Řazení vektorových parametrů do matic [3]

3.3.1 Střední krátkodobá energie

Střední krátkodobá energie [3] se využívá jako detektor řečové aktivity a pro rozlišení hlásek na znělé (vysoká energie) a neznělé (nízká). Často pracujeme s log-energií, která má příznivější (menší) dynamický rozsah. Je nutné si uvědomit, že energie nebude příliš spolehlivá pro detekci řeči a ticha v šumu, především bude selhávat u nízkoenergetických hlásek, které jsou šumem „zamaskovány“. Pro výpočet střední krátkodobé energie u rámce se používá vztah:

$$E = \frac{1}{l_{ram}} \sum_{n=0}^{l_{ram}-1} x^2[n] \tag{4}$$

3.3.2 Počet průchodů nulou

Detekce **průchodů nulou** [3] je dobrá pro rozlišení hlásek na znělé (málo průchodů) a neznělé (více jako šum, tedy více průchodů). Je ale extrémně citlivá na šum a na posuny stejnosměrné složky. Určujeme, kolikrát za rámec projde signál nulou:

$$Z = \frac{1}{2} \sum_{n=1}^{l_{ram}-1} |\sin x[n] - \sin x[n-1]|, \tag{5}$$

kde je $\sin(x)$ zjednodušená znaménková funkce:

$$\sin x[n] = \begin{cases} +1 \;\; pro \;\; x[n] \geq 0 \\ -1 \;\; pro \;\; x[n] < 0 \end{cases} \tag{6}$$

3.4 Fourierova transformace

Na začátku zpracování je signál se spojitým časem, a je definován všude od -∞ do ∞, a čas má ∞ hodnot. Pro reprezentaci signálu ve frekvenční oblasti (spektrum) použijeme **Fourierovu transformaci** [3]:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt, \tag{7}$$

kde funkci $X(f)$ říkáme spektrální funkce, nebo krátce spektrum. Funkce je definována pro $\forall f$ od -∞ do ∞ a je komplexní.

3.5 Cepstrum

Cepstrum [3] slouží pro oddělení buzení a modifikace. V kódování se nám s nimi pracuje lépe samostatně, v rozpoznávání buzení zahazujeme úplně, protože je příliš závislé na řečníkovi (nálada, nemoc, atd.).

Ve frekvenční oblasti je mluvená řeč reprezentována zastoupením jednotlivých frekvencí neboli svým spektrem. Spektrum vzorkovaného signálu je možné spočítat pomocí **Diskrétní Fourierovy transformace (DFT)** [3]:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{nk}{N}} \;\; pro \; k \; \in < 0, N-1 > \tag{8}$$

Samotné spektrum se ale pro klasifikaci rámců nepoužívá. Problémem je, že spektrum, které je získané DFT, stále obsahuje velké množství redundantních informací. Proto se spektrum dále zpracovává a je vypočítáno tzv. cepstrum, které je dáno vztahem:

$$c(n) = DFT^{-1}\{ln|DFT[s(n)]|^2\}, \tag{9}$$

kde $c(n)$ jsou vypočítané cepstrální koeficienty, DFT je diskrétní Fourierova transformace, DFT^{-1} je inverzní DFT a $s(n)$ jsou vstupní vzorky. Výše uvedený postup výpočtu cepstra příliš neodpovídá lidskému slyšení, které má na nízkých frekvencích větší rozlišení než na vysokých (DFT má všude stejné frekvenční rozlišení). Pro rozpoznávače řeči chceme přiblížit cepstrum slyšení, proto se využívají **Mel-frekvenční cepstrální koeficienty (MFCC)** [3].

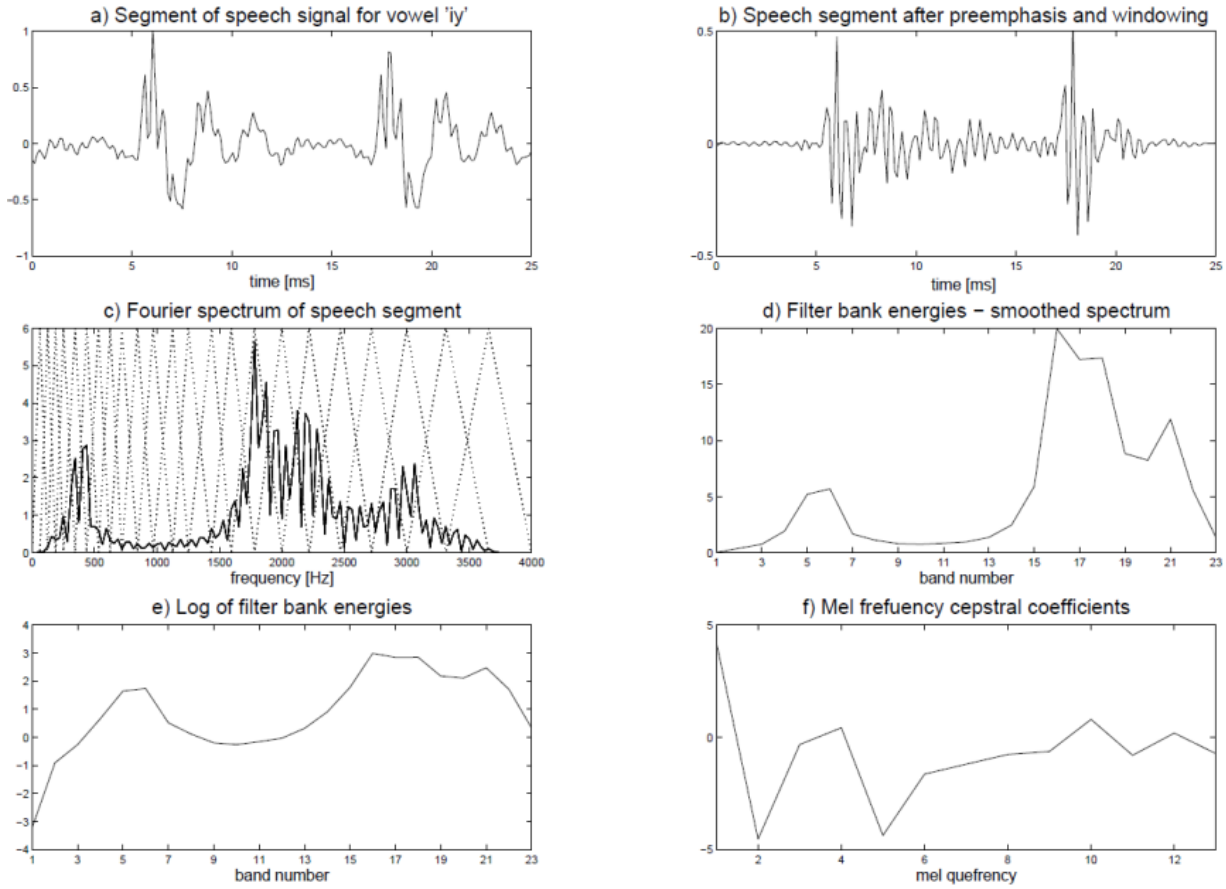
MFCC se snaží kompenzovat zejména nelineární vnímání frekvencí, a to s využitím banky trojúhelníkových filtrů s lineárním rozložením frekvencí v tzv. melovské frekvenční škále, která je definována vztahem (převod Hertzů na Mely):

$$F_{Mel} = 2959 \log_{10}(1 + \frac{F_{Hz}}{700}), \tag{10}$$

Potom jsou energie získané z jednotlivých pásem zlogaritmovány a vektor energii je transformován do cepstralní oblasti. Pro výpočet cepstralních koeficientů se nepoužívá Fourierova transformace, ale je využita **diskrétní Cosinova transformace (DCT)** [3]. Vztah pro výpočet Mel-frekvenčních cepstrálních koeficientů je uveden zde:

$$c_{mf}(n) = \sum_{i=1}^{K} \log m_k \cos\left[n(k - 0.5)\frac{\pi}{K}\right] \tag{11}$$

Další možností jak dostat lepší výsledky je použití **lineární predikce cepstrálních koeficientů (LPCC)**[3]. Níže (obrázek 5) jsou zobrazeny jednotlivé signály při výpočtu MFCC.



Obrázek 5: Výpočet MFCC: a) původní signál, b) preemfázovaný signál, c) DFT spektrum a jeho váhování filtry, d) energie na výstupech jednotlivých filtrů, e) log této energie, f) MFCC [3]

4 Rozpoznávání řeči

Úkolem pro rozpoznávání řeči (speech recognition) je zjistit, co bylo řečeno. Rozpoznávání je citlivé na rozdíly v intonaci i výslovnosti a může být jednak **závislé na mluvčím** [1] **(SD - Speaker Dependent)** či na **mluvčím nezávislé** [1] **(SI - Speaker Independent).**

Systémy, které jsou na mluvčím závislé, dosahují pro danou osobu lepších výsledků, ale jejich nevýhodou je, že mluvčí musí udělat několik nahrávek, aby bylo možné natrénovat dobré modely.

Systémy, které jsou na mluvčím nezávislé, byly natrénované pomocí nahrávek od velkého množství lidí. Nedosahují ovšem takové úspěšnosti jako systémy SD. Bylo proto vyvinuto několik metod, které SI model adaptují na konkrétního mluvčího. Mezi základní metody adaptace patří metody **MAP** [5] **(maximum a posteriori estimation – maximální aposteriorní pravděpodobnost)**, **MLLR** [5] **(maximum likelihood linear regression – maximální věrohodná lineární regrese)** a jejich různé variace. Podle složitosti klasifikujeme systémy pro rozpoznávání do těchto kategorií [3]:

- **izolovaná slova** – ovládání mobilních telefonů hlasem, potřebují voice aktivity detector nebo push-to-talk
- **spojená slova (omezený slovník)** – rozpoznávání číslovek při zadávání telefonního čísla nebo čísel kreditních karet. Rozpoznávání je většinou řízeno nějakou sítí nebo jednoduchou gramatikou.
- **plynulá řeč s velkým slovníkem** – je to nejtěžší úkol, potřebuje informace o akustice, ale také o struktuře jazyka (jazykový model) a výslovnostní slovník. Pracují s menšími jednotkami než se slovy (60 tisíc slov se nedá naučit). Využívají fonémy a kontextově závislé fonémy.

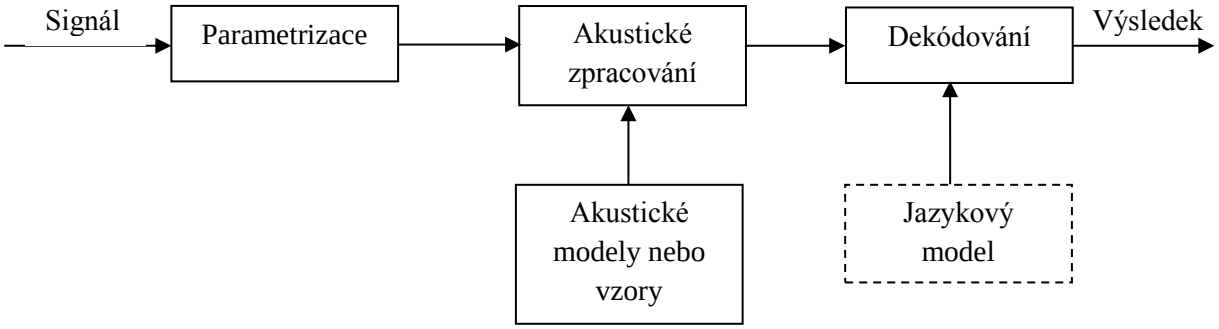
Starší, dnes již téměř nevyužívaná metoda, je metoda **dynamického borcení času** [1] **(DTW).** Tato metoda je vhodná pro rozpoznávání izolovaných slov (například pro hlasové ovládání počítače), protože je zapotřebí mít ve slovníku uloženou modelovou nahrávku každého použitého slova. Metoda měří podobnost mezi dvěma nahrávkami slov (popřípadě slovních spojení) a snaží se najít nejlepší shodu. U sekvencí, které se porovnávají, je třeba brát v potaz jejich rychlost a délku (musí se vyrovnávat).

Moderní univerzální systémy rozpoznávání řeči jsou založeny na **skrytých Markovských modelech** [1] **(HMM)**. Jedná se o statistické modely, které obsahují skryté stavy. HMM se používají v rozpoznávání řeči, protože řečový signál může být chápán jako po částech stacionární signál, nebo po krátkých časových úsecích stacionární signál. V krátkém čase (např. 10 ms), může být řeč aproximována jako stacionární proces. Dalším důvodem proč jsou HMM populární je, že mohou být natrénovány automaticky a jsou jednoduché a výpočetně vhodné pro použití. Tato metoda se používá pro spojité rozpoznávání řeči, což je mnohem komplexnější a náročnější úloha než rozpoznávání izolovaných slov. Při vyslovování izolovaných slov se mluvčí více soustředí na konkrétní slovo a vyslovuje jej zřetelněji. Při spojité řeči dochází také k některým jevům jako je například spodoba slov (poslední hláska ve slově je ovlivněna následující hláskou, kterou začíná druhé slovo), polykání hlásek, apod. Pro rozpoznávání se často používá **Viterbiho algoritmus** [5].

Jako atraktivní přístup v rozpoznávání řeči se ukázalo využití **neuronových sítí** [1]. Tato metoda se využívá u klasifikace fonémů, rozpoznávání izolovaných slov a adaptace mluvčího. Neuronové sítě je také možné použít na předzpracování signálu pro HMM rozpoznávání řeči.

4.1 Struktura rozpoznávače

Na obrázku je znázorněna typická struktura rozpoznávače. Na vstupu přichází již předzpracovaný signál, na který je provedena parametrizace. Ta má za úkol omezit množství dat, které získáváme z řeči, a vyjmout složky, které nás nezajímají. Parametrizace často používá MFCC a LPCC. Výsledkem je potom sekvence vektorů.



Obrázek 6: Typická struktura rozpoznávače

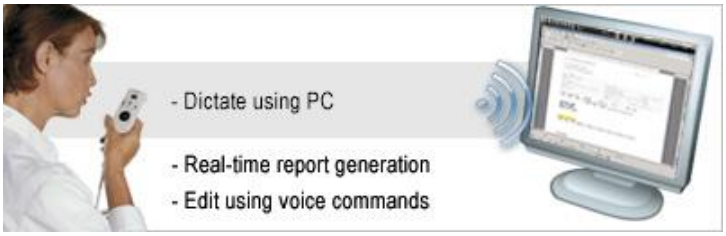
Akustické zpracování [3] má za úkol vypořádat se dvěma zdroji variability, k čemu se využívají akustické modely a vzory. Člověk totiž neřekne nikdy jednu věc úplně stejně, vektory parametrů se proto vždy liší (nálada řečníka, zdravotní stav). S tímto se můžeme vyrovnat **měřením vzdálenosti mezi vektory** [3] nebo **statistickým modelováním** [3]. Další příčinou variability je odchylka v čase. Opět není možné říkat jednu věc se stejným časováním. S tímto případem se můžeme vyrovnat použitím srovnávacích cest nebo stochastického automatu.

Poslední fází je dekódování. Jedná-li se o rozpoznávání izolovaných slov, tak je to velmi jednoduché. Budeme pouze vyhledávat maximum pravděpodobnosti nebo minimum vzdálenosti. U plynulé řeči je zapotřebí pracovat se složitějšími akustickými modely, jazykovým modelem a výslovnostními slovníky.

5 Využití rozpoznávačů

V této kapitole se dočtete, kde lze rozpoznávače řeči uplatnit a jaké firmy se na vývoji rozpoznávačů řeči podílejí. Nejsou zde uvedeny detaily implementací jednotlivých řešení, ale pouze jejich popis.

V praxi rozeznáváme dva druhy rozpoznávání. **Přímé rozpoznání řeči** [6] je prováděno v reálném čase, zatímco autor diktuje. Používá se pro krátké zprávy, nebo když není dostatek transkripčních zdrojů. Pomocí hlasových příkazů může uživatel procházet, upravovat či formátovat svůj diktát.



Obrázek 7: Přímé zpracování řeči [6]

Dalším typem je **zpracování řeči pomocí serveru** [6]. Uživatel nahrává řeč pomocí nějakého zařízení, které může zvuk ještě nějak modifikovat, a potom ho pošle na server, kde proběhne rozpoznávání. Následně proběhnou korekce a výsledek se pošle zpět uživateli. Ten si potom může výsledek upravit dle svých požadavků.

Obrázek 8: Zpracování řeči pomocí serveru [6]

Ve zdravotnictví, soudnictví, policejních složkách a jiných organizacích, kde je zapotřebí dělat nějaké záznamy se rozpoznávání řeči využívá pro přepis diktátů. Jedná se o zpracování plynulé řeči, která navíc potřebuje konkrétní slovník. Jazyk každého oboru se liší slovní zásobou, ustálenými způsoby stavby vět a frází apod. Jazykové modely jsou vytvářeny ze skutečných textů (např. soudních spisů, lékařských zpráv a nálezů, článků novin a časopisů), které se v daném oboru vyskytují v jeho reálném prostředí. Tyto programy zajistí i takové úpravy rozpoznaného textu, které se řídí pravidly pravopisu (např. při psaní velkých a malých písmen, sledování gramatické shody podmětu a přísudku, shody v rodech přídavných jmen, mezery mezi slovy apod.). Tyto programy neslouží pouze k usnadnění práce, ale pomáhají lidem s postižením, které jim nedovolí psát.

Systémy pro přepis diktátů vyvíjí firmy **Crescendo Systems** [6] a **SpeachTech** [7]. U společnosti SpechTech se jedná konkrétně o aplikaci **MegaWord** [7], která využívá nejnovějších výsledků výzkumného projektu MegaWord.cz, který probíhá za spolupráce se Západočeskou univerzitou v Plzni. Jednou z brandovaných variant produktu MegaWord je software **NovaVoice®**[8], který je distribuován společností Consulting Company Novasoft a.s. Tento diktovací systém se chlubí až 97% úspěšností a není závislý na mluvčím. V případě potřeby je ale možné akustické modely přizpůsobit konkrétnímu řečníkovi (třeba vada v řeči).

Armáda využívá systémy pro rozpoznání řeči třeba ve stíhacích letounech, kde mají pomáhat pilotovi s ovládáním různých kokpitových funkcí (žádné kritické systémy) a zlepšit tak jeho koncentraci a pozornost při řízení. Podobné systémy se používají třeba i u helikoptér. Největším problémem je odfiltrovat akustický hluk a rozpoznat pouze řeč. Výzkumu se účastní U. S. Army Avionics Research and Development Activity (AVRADA) a Royal Aerospace Establishment (RAE) v Anglii.

U mobilních telefonů se dnes již setkáváme s inteligentním mobilním asistentem, který nám má ulehčit práci. Jedná se o systémy využívající rozpoznávač řeči, které spolupracují s dalšími aplikacemi v telefonu. Tito asistenti nám dokáží odpovídat na otázky (kolik je hodin, jaké je počasí atd.), mohou ovládat mobilní zařízení, vyhledávat na webu či mapách, nastavují GPS a mnoho dalšího.

Jednou z prvních takovýchto aplikací byla **Siri** [9] **(Speech Interpretation and Recognition Interface)**, která byla integrována na zařízení iPhone 4S 4. října 2011. Využívá řečové technologie společnosti **Nuance** [10]. Tato firma se mimo jiné zabývá integrací rozpoznávačů či syntetizátorů řeči do dalších aplikací (Dragon Dictation, T9, Swype, atd.). Siri Musí komunikovat se serverem, který provádí vlastní rozpoznávání, proto je potřeba být připojen k internetu.

Společnost Google nezůstala pozadu a 27. července 2012 představila svého inteligentního asistenta **Google Now** [11], který je už na Androidu 4.1. Na rozdíl od Siri nepotřebuje server, ale je to aplikace běžící pouze na zařízení. Díky tomu je trochu rychlejší, na druhou stranu potřebuje taky více místa a výpočetní kapacity. Dříve než vydali mobilního asistenta, tak už využívali rozpoznávání řeči ve svém vyhledávači, mapách a na YouTubu. Dnes zde existuje podpora hodně světových jazyků a to i včetně češtiny.

V roce 2007 Microsoft odkoupil společnost Tellme Netwoks a tím vzniklo oddělení **Microsoft TellMe** [12], které se zabývá zpracováním řeči (i syntetizace). Zákazníci mohou vytvářet internetové aplikace běžící na Tellme síti a využívající Tellme Studio, což je webový vývojový nástroj. Tato platforma je založena na otevřených standardech jako VoiceXML, CCXML a VoIP. Microsoft využívá rozpoznávače na Windows 7, herních konzolí Xbox a na mobilních platformách Windows Phone 7. Microsoft TellMe se dále učí od uživatelů pomocí tzv. „Cloud platformy“, kde jsou zaznamenány dotazy od uživatelů. Spolupracuje také s firmou Ford (Ford SYNC) a KIA (KIA UVO).

Na novém zařízení Samsung Galaxy S3 (a dalších novinek) je vlastní asistent **S Voice** [13]. Komunikuje také se serverem a ve funkcích je velice podobný ostatním. Samsung mimo jiné vyrábí i inteligentní televizory, které mimo pohybových senzorů a kamer obsahují i systém pro zpracování řeči. Díky nim je možné ovládat televizory pomocí hlasových příkazů.

Žádný mobilní asistent zatím nepodporuje český jazyk, proto je obtížné např. vyhledávat v mapách a používat tak třeba GPS. Kromě těchto odvětví se řeč využívá třeba ve vesmírných projektech. NASA v projektu Mars Polar Lander využívala rozpoznávače společnosti Sensory, Inc. V automobilovém průmyslu se může pomocí hlasu ovládat hands-free, navigace, atd.

6 Android

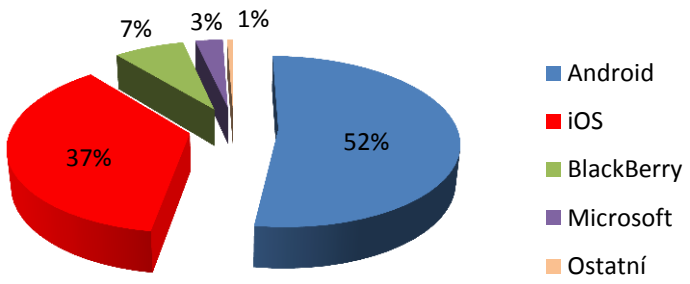
Softwarová platforma Android spatřila světlo světa 5. listopadu 2007. Tohoto dne byla oficiálně ohlášena a zároveň s ní vzniklo sdružení firem Open Handset Alliance (OHA), pod které spadá celkem 34 výrobců hardwaru a softwaru a telekomunikačních společností. Téhož dne byla platforma Android předána Googlem právě tomuto sdružení. Nicméně tento nápad vnikl už v roce 2003, kdy se společnost Android Inc. zabývala vývojem aplikací pro tzv. chytré telefony. Americká společnost Google, která působila převážně v oblasti internetových aplikací a vyhledávání, se při stoupající oblibě a prodeje chytrých multimediálních telefonů rozhodla v roce 2005 tuto firmu odkoupit. Vývoj tohoto nového systému nebyl ohlášen, dalo by se říci, že byl utajován a na konec vznikla nová platforma - Android, která se měla stát konkurentem již existujícím platformám Apple iOS, Symbian a Windows Mobile.

Android je tedy rozsáhlá open-source platforma, která je určena zejména pro mobilní zařízení (chytré telefony, PDA, navigace, tablety). Obsahuje v sobě operační systém (založený na jádře Linuxu), middleware, uživatelské rozhraní a aplikace. Při vývoji systému byla brána v úvahu technická omezení, kterými by mohli mobilní zařízení disponovat jako výdrž baterie, menší výkonnost a málo dostupné paměti. Některá tyto omezení již v dnešní době ale neplatí např.: čtyř jádrové procesory (1,9 GHz), 2 GB RAM a vlastní grafické procesory. Dále bylo jádro Androidu navrženo pro běh na různém hardwaru. Systém lze tak použít na různých zařízeních bez ohledu na chipset, velikost či rozlišení obrazovky. Platforma Android se dodává s kompletním řešením nasazení operačního systému (specifikace driverů, základní aplikace aj.).

V říjnu roku 2008 byl uveden na trh ve Spojených státech telefon vyrobený firmou HTC s operačním systémem Android. Do České republiky dorazil v lednu 2009. Zároveň s tím bylo pro vývojáře uvolněno SDK 1.0 [17][19].

6.1 Podíl na trhu

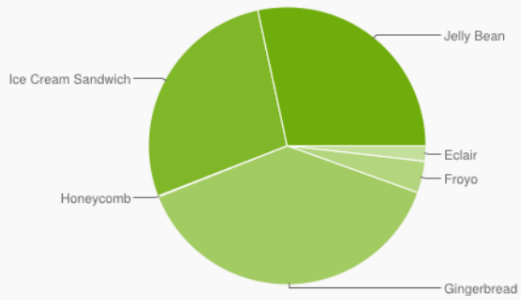
Platformu Android jsem si vybral také díky její stoupající oblibě a v této kapitole bych chtěl ukázat jak časté je dnes její využití. Společnost ComScore zabývající se výzkumem trhu zveřejnila výsledky na poli chytrých telefonů. Výsledky jsou znázorněny níže a to ke konci roku 2012 [14].



Graf 1: Podíl na trhu k 31. 12. 2012

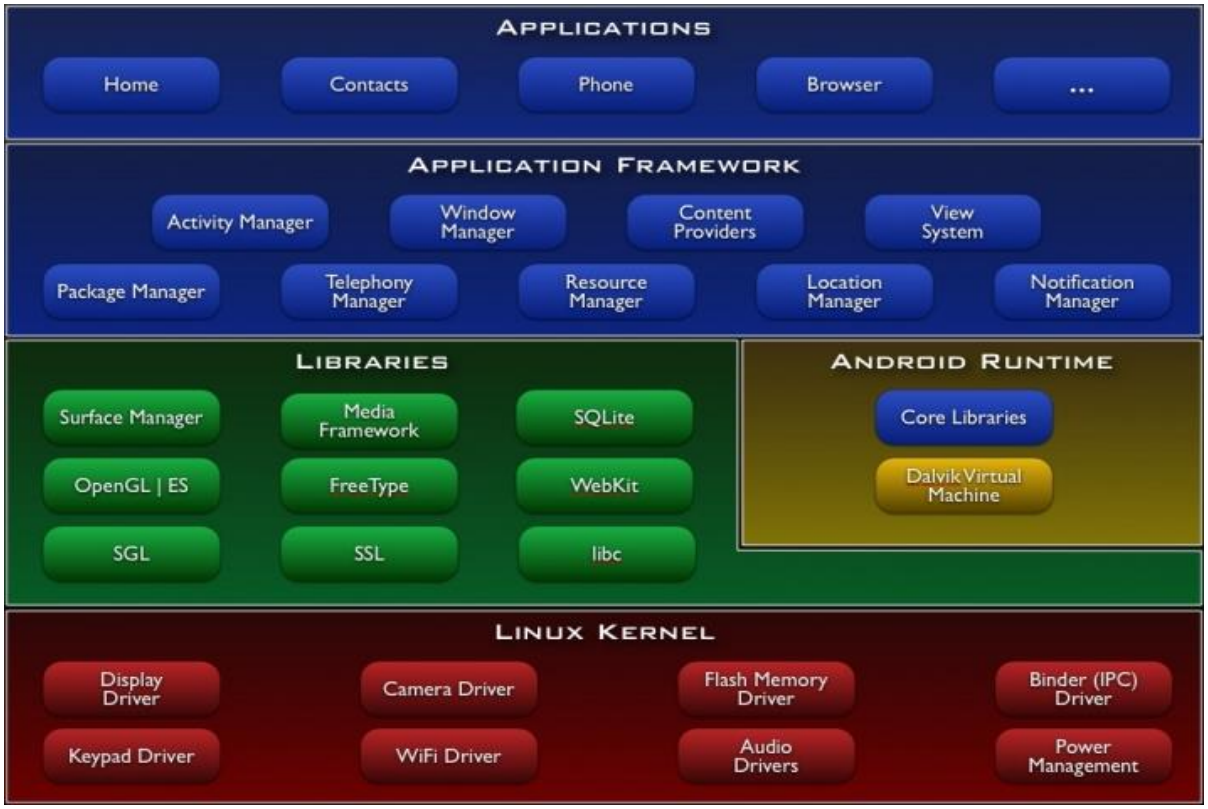
Google navíc vydává každých 14 dní procentuální zastoupení jednotlivých verzí systému, které běží na mobilních zařízení. Jak je patrné z tabulky, více jak třetina zařízení uživatelů má stále verzi 2.3, a právě proto jsem se snažil při implementaci využívat funkce z co nejnižších verzí API. Podařilo se mi vyvinout plně funkční aplikaci využívající API 4, což je Android 1.6.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	3.7%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	38.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
4.1.x	Jelly Bean	16	26.1%
4.2.x		17	2.3%



Obrázek 9: Verze systému Android k datu 1. 5. 2013 [15]

6.2 Architektura platformy Android



Obrázek 10: Hlavní komponenty architektury platformy Android [16]

Platformu android lze rozdělit do pěti vrstev (viz obrázek) [17][18]:

- **Linuxové jádro (Linux Kernel)** – Jedná se o nejnižší vrstvu této architektury, která tvoří abstraktní vrstvu mezi používaným hardwarem a ostatními vrstvami. Starší verze Androidu (do 4.0) byli postaveny na Linuxovém jádře 2.6, od Androidu 4.0 bylo použito jádro verze 3.0.1. Tato vrstva má mimo jiné za úkol i správu paměti, sítí, procesů, napájení, bezpečnosti a obsahuje ovladače hardwarových komponent.
- **Knihovny (Libraries)** – Knihovny jsou napsány v jazyce C/C++ a využívají je různé komponenty. Vývojářům jsou poskytnuty prostřednictvím Android Application Framework. Toto jsou některé knihovny systému Android:
    - **Systém C Library (libc)** – Implementace standartní systémové knihovny C odvozená od BSD implementace a upravená pro linuxová zařízení.
    - **Media Libraries** – Knihovna pro přehrávaní a nahrávání video a audio formátů, zobrazení obrázků (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG).
    - **Surface Manager** – Správa zobrazení displeje, spojuje 2D a 3D grafické vrstvy a umožňuje vytváření grafických efektů (animace, průhlednost, posuny, rotace, atd.).
    - **SGL** – Základní knihovna pro 2D grafiku.
    - **OpenGL ES** – Knihovna pro vykreslování 3D grafiky.
    - **FreeType** – Knihovna pro rendering bitmapových a vektorových fontů.
    - **WebKit** – Knihovna webového prohlížeče, který podporuje i vložené náhledy webových stránek.
    - **SSL** – Knihovna pro zabezpečenou síťovou komunikaci.
    - **SQLite** – Odlehčená relační databáze.
- **Android Runtime** – Tato vrstva se stará o chod aplikací psaných v jazyku Java na virtuálním stroji zvaném Dalvik. Obsahuje tedy:
    - **Core Libraries** – Základní programovací knihovny jazyka Java, které obsahově téměř odpovídají platformě Java Standart Edition. Hlavní rozdíl je nahrazení knihoven pro uživatelské rozhraní (AWT a Swing) knihovnami optimalizovanými pro rozhraní Android a jsou zde přidány knihovny Apache pro práci se sítí.
    - **Dalvik Virtual Machine (DVM)** – Jedná se o aplikační virtuální stroj s registrově orientovanou architekturou, který byl vyvíjen od roku 2005 speciálně pro Android. Využívá základních vlastností Linuxového jádra, jako je správa paměti nebo práce s vlákny. Jeho vznik podnítili dva hlavní důvody. Prvním důvodem byla licenční práva na jazyk Java, který je volně šiřitelný, zatím co Java Virtual Machine (JVM) není. Druhým důvodem byla optimalizace virtuálního stroje pro mobilní zařízení, kde se musí brát v potaz ohled na poměr úspory energie a výkonu.
- **Aplikační Framework (Application Framework)** – Vrstva, která je pro vývojáře asi nejdůležitější. Umožnuje programátorovi využít možnosti operačního systému Android přes aplikační rozhraní, kterému se říká API (Application programming interface). Všechny aplikace včetně těch základních využívají stejné API, což umožňuje jednotlivé aplikace nahradit vlastními (skoro každý výrobce mobilních zařízení s Androidem si ho optimalizuje podle svých představ a požadavků). Navíc je u Androidu možné opakované použití

jednotlivých komponent. Jakákoliv aplikace může být tedy využívána jinou aplikací (záleží na bezpečnostních omezeních frameworku). Základní sada služeb zahrnuje především:

- **Systém pohledů (View System) –** Obsahuje prvky pro sestavení uživatelského rozhraní. Dají se použít již předefinované prvky, jako jsou tlačítka, textové pole, seznamy, checkboxy, posuvníky a jiné.
- **Dodavatel obsahu (Content Provider) –** Umožňuje přístup k obsahu jiných aplikací (např. kontakty, fotky, zprávy atd.).
- **Správce prostředků (Resource Manager) –** Poskytuje přístup ke zdrojům, které nejsou součástí aplikačního kódu, jako jsou řetězce, grafika, přidané soubory.
- **Správce upozornění (Notification Manager) –** Umožňuje všem aplikacím zobrazit vlastní upozornění ve stavovém řádku, přehráním zvuku, rozsvícením LED diody, vibracemi.
- **Správce aktivit (Activity Manager) –** Řídí životní cyklus aplikací a poskytuje orientaci v zásobníku s aplikacemi.

- **Aplikace (Applications)** – Jedná se o nejvyšší vrstvu systému, kterou využívají běžní uživatelé. Jedná se o aplikace již nainstalované a nezbytné pro správný chod zařízení i o aplikace stažené a nainstalované z Android Marketu.

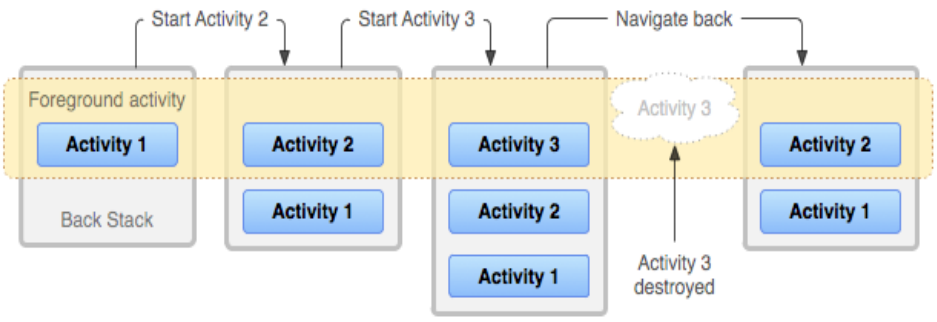
6.3 Komponenty aplikace

Jedná se o základní stavební prvky aplikace pro Android. Některé komponenty jsou na sobě závislé, ale každá existuje jako vlastní prvek, který pomáhá definovat celkové chování naší aplikace. Všechny tyto komponenty musí být definovány v souboru AndroidManifest.xml. Mohou mezi sebou komunikovat pomocí zpráv, tzv. *intentů*. Specifikace chování komponent na přijatou zprávu je určena pomočí tzv. *intent filtrů*. Mezi čtyři základní komponenty patří *aktivity* reprezentující obrazovky, *services* umožňující provádět akce na pozadí, *contetnt providers* poskytující přístup k datům a nakonec *broadcast receiver* reagující na příchozí zprávy [17][18][19][20].

6.3.1 Aktivity (Activity)

Aktivita představuje jednu obrazovku s uživatelským rozhraním, se kterým mohou uživatelé něco dělat (vytočit číslo, napsat zprávu, atd.). Novou aktivitu definujeme jako podtřídu třídy Activity. Přestože aktivity pracují společně na splnění požadovaného úkolu, je každá aktivita nezávislá na ostatních. Aplikace je tedy většinou složena z několika aktivit, které jsou vázány k sobě navzájem.

Obvykle je jedna aktivita zvolena jako hlavní, a ta nabíhá při prvním spuštění aplikace. Každá aktivita potom může spouštět další jiné aktivity, které nám rozšiřují funkčnost aplikace. Pokaždé když je spuštěna nová aktivita, se předešlá zastaví, ale systém ji zachová v zásobníku (back stack). Pokud tedy přepneme z jedné aktivity na jinou, tak se nám ta nová aktivita přidá na vrchol zásobníku a přejde do režimu focus. Po jejím skončení (zmáčknutí tlačítka back) se ze zásobníku vybere (a zničí) a označí se za aktivní ta za ní. Na následujícím obrázku je znázorněna funkce zásobníku [19][21].



Obrázek 11: Funkce zásobníku aktivit (back stack) [22]

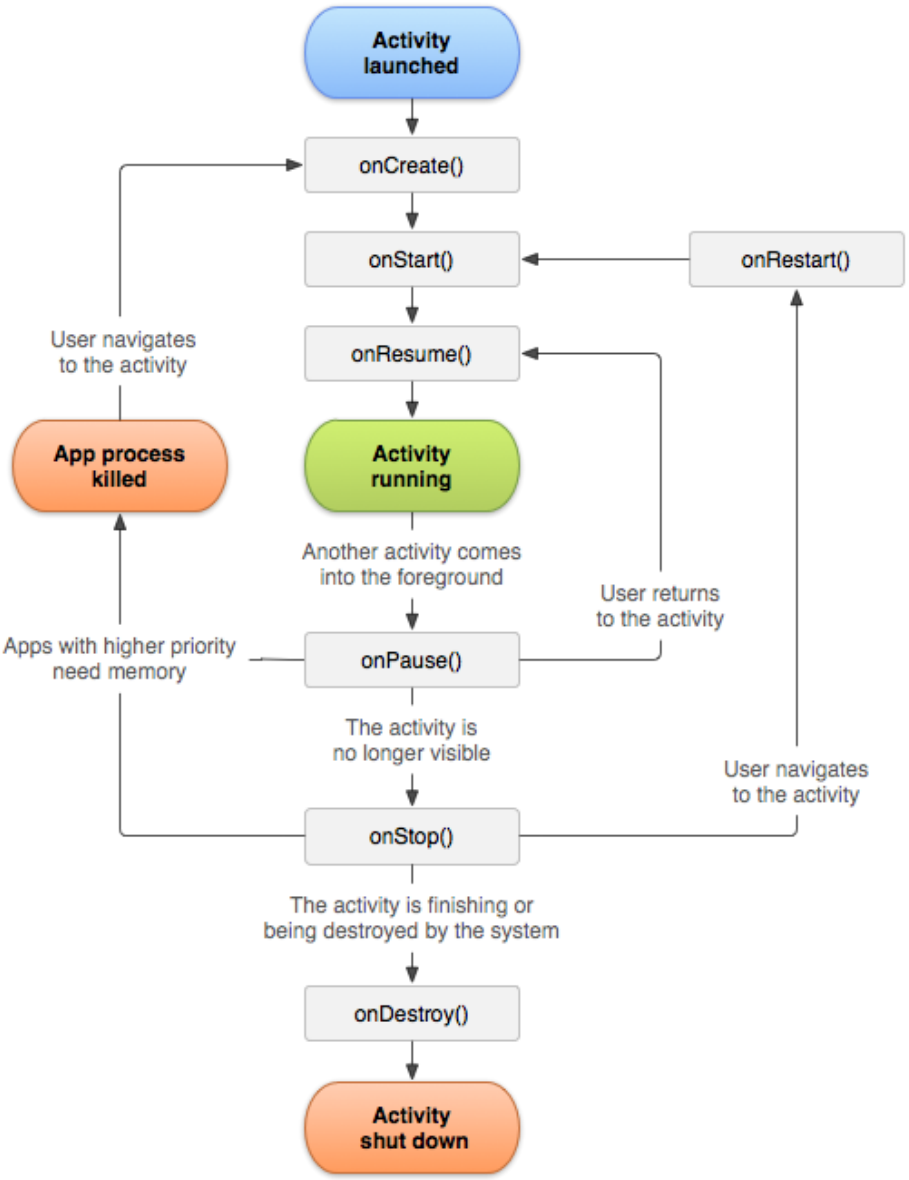
Zahájení nové aktivity je poměrně náročná záležitost. Musí se vytvořit nový proces, alokovat paměť pro objekty uživatelských rozhraní a vyvolat zobrazení. Aby nedocházelo ke zbytečnému plýtvání výpočetních prostředků je zde Activity Manager, který zodpovídá za vytváření, rušení a celkovou správu životního cyklu aktivit. Životní cyklus aktivity se může nacházet v těchto stavech [19][21]:

- **Activity launched** – Inicializace aktivity.
- **Activity running** – Aktivita běží a je zobrazena na displeji (má focus). Může tedy komunikovat s uživatelem. V jediném okamžiku může být v tomto stavu právě jedna aktivita.
- **App process killed** – Activity Manager zrušil aktivitu z důvodu nedostatku paměti. K této akci dochází, pouze pokud není aktivita viditelná.
- **Activity shut down** – Activity Manager ukončil aktivitu a ta již nevyužívá žádné prostředky či paměť.)

6.3.2 Dodavatelé obsahu (Content provider)

Slouží jako aplikační rozhraní pro sdílení dat mezi aplikacemi, ale i pro sdílení dat mezi jednotlivými aktivitami jedné aplikace. Pro správu dat slouží třída *ContentResolver*. Komunikace mezi dodavateli obsahu a aplikací je typu klient-server, kde *ContentResolver* vystupuje v roli serveru. Oddělení dat od uživatelského rozhraní nabízí možnost nahrazení výchozích aplikací novými. Můžeme třeba použít uložené kontakty, zprávy a multimédia. Možnosti uložení dat jsou následující [20][26]:

- **Interní úložiště** – Data jsou uložena ve vnitřní paměti zařízení a většinou nejsou přístupná ostatním aplikacím. Po odinstalování nějaké aplikace jsou smazána i její data.
- **Externí úložiště** – Data jsou uložena v externí paměti. V současnosti se jedná především o SD karty. Je možné ukládat i data aplikací a rozšiřuje to tedy interní paměť.
- **Shared Preferences** – Ukládá se zde pouze malé množství dat a je možné použít pouze základní datové typy (integer, string, boolean, …). Ukládání ve formě klíč-hodnota. Tento způsob by se dal přirovnat k cookies a využívá se třeba pro uložení nastavení aplikace.
- **SQLite** – Odlehčená relační databáze, která je přístupná pouze v rámci aplikace.
- **Síťové úložiště** – Využití vlastního serveru nebo cloudu pro uchování dat.



Obrázek 12: Životní cyklus aktivity [23]

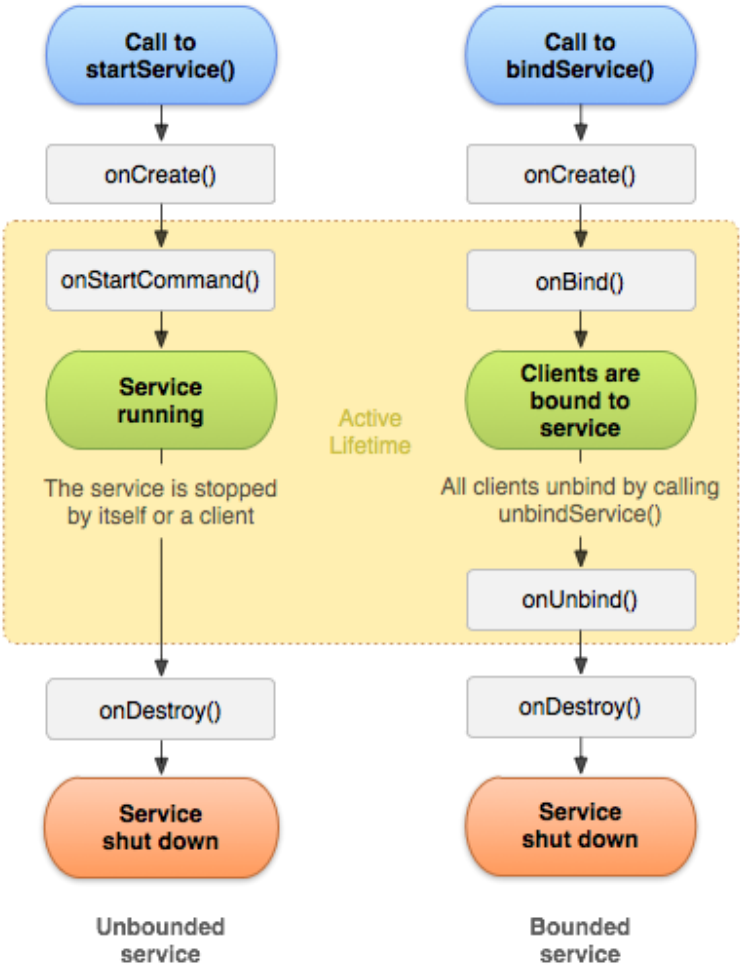
6.3.3 Služby (Services)

Služba je komponenta, která může provádět dlouhotrvající operace na pozadí bez interakce s uživatelem. Proto neposkytuje uživatelské rozhraní a má vlastní životní cyklus. V aplikaci může každá komponenta spustit službu a ta poběží na pozadí i v případě, že uživatel přepne na jinou aplikaci. Každá komponenta může kromě spuštění služby využívat i služby již běžící (přivázat se ke službě) a dokonce provádět meziprocesovou komunikaci (IPC). Každá služba může být spuštěna pouze jednou a to, i když ji používá více aktivit. Využívají se například při příjmu síťových transakcí, přehrávání hudby na pozadí a provádění I/O operací se soubory. Služby lze rozdělit v zásadě na dvě formy [20][24]:

- **Spuštěná (started)** – Služba je spuštěna z jiné komponenty metodou *startService()*. Poté co je služba spuštěna může běžet na pozadí neurčitou dobu a to, i když je komponenta, která ji spustila, zastavena. Obvykle vykonává pouze jednu operaci a výsledek vrátí tomu, kdo ji

spustil. Po dokončení činnosti se sama zastaví metodou *stopSelf()* nebo ji nějaká komponenta ukončí zavolání funkce *stopService()*.

- **Vázaná (bound)** – Druhým způsobem je připojení ke službě metodou *bindService()*. Jestliže služba neexistuje, tak ji vytvoří první připojující se klient. Klient může se službou komunikovat pomocí rozhraní *IBinder* nebo meziprocesovou komunikací – IPC. Službu může využívat více komponent naráz a lze se odpojit metodou *unbindService()*. Po odpojení všech klientů je služba zastavena.



Obrázek 13: Životní cyklus služby [25]

6.3.4 Přijímače vysílání (Broadcast receiver)

Broadcast reciever je komponenta, která se používá k čekání (poslouchání) broadcastového oznámení. Podle určení na ně reaguje, například výpisem na stavový řádek nebo spuštěním jiné komponenty. Aplikace mohou využívat tyto přijímače buď systémové, nebo umí vytvářet své vlastní. Podobně jako služby ani broadcast receiver nemá uživatelské rozhraní. Při přijetí nějaké zprávy se zavolá metoda *onReceive()*. Proces běžícího receiveru nemůže být zastaven a ani ukončen. Pro příliš časově náročné operace je možné pro jejich zpracování spustit službu. Příklad použití broadcast recieveru může být

reakce na oznámení o nízkém stavu baterie, o zachycení fotografie, doručení SMS zprávy nebo stažení dat [26].

6.4 Vývoj aplikací

Aplikace jsou pro platformu Android psány v jazyce Java. Pro programy jejichž odezva je kriticky důležitá je možné použít jazyk C/C++. Pro distribuci jednotlivých aplikací slouží formát Android package (koncovka *.apk*), který je ve skutečnosti pouze archiv rozbalitelný programem ZIP. Archiv obsahuje zkompilovaný Java kód a všechny ostatní potřebná data a soubory [17].

Vývoj aplikací na této platformě představuje jedinečný koncept. Jednotlivé aplikace jsou složeny z komponent a každá aplikace může využívat komponenty i z jiných aplikací. Navíc tvorba programů pro mobilní zařízení je odlišná od programování na PC. Aplikace musí být přizpůsobena malému displeji a musí si současně zachovat i její funkčnost. Navíc musíme brát v potaz i jistá omezení jako jsou výdrž baterie nebo připojení k internetu (uživatel omezen FUP, rychlostí či cenou) [31].

6.4.1 SDK

Při vývoji aplikací pro systém Android je potřeba si v prvním kroku stáhnout a nainstalovat balíček Android SDK (Standart Development Kit), který je dostupný z oficiálních stránek Androida pro vývojáře. Tento balíček obsahuje základní nástroje a některé API potřebné k vytváření aplikací. Navíc obsahuje debugger, knihovny, dokumentaci, ukázkové příklady, tutoriály a emulátor mobilních zařízení pro testování. Instalátor je možné si stáhnout pro Windows, Linux i MAC OS. Po instalaci balíčku, který obsahuje jen základ, si lze pomocí SDK manažera najít a doinstalovat další knihovny, tutoriály a ostatní verze API. Tady je stručný výčet nástrojů SDK [27][28]:

- **Android Virtual Devices (AVD)** – Správce emulátorů. Pomocí něho lze vytvářet, mazat a měnit virtuální zařízení. Počet zařízení není omezen a lze je vytvářet se specifickým nastavením (paměť, displej, API, …)
- **Davlik Debug Monitor Service (DDMS)** – Díky této službě lze na připojených či emulovaných mobilních zařízení spravovat procesy. To se využívá zejména při ladění aplikace. Můžeme sledovat a ukončovat procesy, logovat chyby, sledovat data a vlákna nebo pořizovat snímky a videa obrazovky.
- **Android Debug Bridge (adb)** – Umí přistupovat do emulátorů nebo připojených zařízení pomocí příkazového řádku. Umožňuje ladit aplikace a nahrávat soubory a balíčky.
- **Android Asset Packaging Tool (aapt)** – Vytváří a modifikuje výsledné aplikační balíčky.
- **Android Interface Description Language (aidl)** – Vytváří rozhraní pro komunikaci mezi procesy pomocí jazyka IDL. Díky tomuto nástroji si mohou procesy a služby v systému předávat data pomocí meziprocesové komunikace – IPC.
- **Draw 9-path** – Nástroj pro tvorbu NinePath grafiky. Slouží pro tvorbu roztahovatelných prvků, jako jsou třeba tlačítka. Jedná se bitmapovou grafiku.

- **Hierarchy Viever** – Umožňuje vytvářet a ladit uživatelská rozhraní. Z xml souboru, kde je definováno GUI dokáže zobrazit stromovou strukturu jednotlivých prvků. Nebo může GUI zobrazit v režimu Pixel Perfect View, které nám *xml* soubor převede do grafické podoby a lze jej pomocí editoru upravovat.
- **Sqlite3** – Aplikace pro přístup do SQLite databáze.
- **Mksdcard** – nástroj pro emulaci paměťových karet.
- **Dx** – Překladač Java souborů do spustitelných *dex* (Dalvik Executable Format) souborů.

6.4.2 ADT

Android Development Tools (ADT) je oficiální plugin pro vývojové prostředí Eclipse, který je navržen tak, aby poskytl co nejlepší prostředí pro tvorbu Android aplikací. Díky němu lze pohodlně ovládat celé SDK. Součástí tohoto pluginu je snadné vytváření Android projektů, jejich překlad a instalace na připojené zařízení. Pomocí ADT lze také instalovat emulátory přímo v Eclipse a program spouštět na vytvořeném zařízení. Tento postup mohou využít programátoři, kteří nevlastní mobilní zařízení s Androidem nebo potřebují program otestovat s určitými parametry zařízení, které lze nastavit (verze API, velikost displeje, atd.). Dalšími výhody ADT jsou vestavěný debuger DDMS, editor XML dokumentů, nástroj pro tvorbu a editaci GUI a integrovaná dokumentace SDK přímo v prostředí Eclipse (seznamy tříd, metod, jejich popis, atd.) [29].

6.4.3 NDK

Native Development Kit (NDK) je sada nástrojů, která nám umožňuje implementovat části aplikace v nativním kódu v jazycích C a C++. Tato metoda je vhodná v částech, kde je požadována výkonost a rychlost zpracování. Jednou z výhod použití nativních kódů je možnost použití již napsaných knihoven či programů v jazyce C/C++. Vývojář by měl vždy zvážit použití těchto kódů, protože kromě zlepšení výkonosti to s sebou přináší i zvýšení složitosti aplikace a nepřehlednosti kódu. Dobrými kandidáty pro použití NDK jsou algoritmy intenzivně využívající procesor, které nepotřebují moc paměti, jako je zpracování signálu, simulace fyziky, kódování, atd. K provozu NDK je zapotřebí SDK s API 3 což odpovídá Androidu verzi 1.5.

Podobně jako u SDK je možné si NDK stáhnout a naistalovat na systémy Windows, Linux i MAC OS. Obsahuje potřebné API, ukázkové příklady, kompilátory, linkery, atd.. Navíc poskytuje množinu systémových hlaviček pro stabilní nativní rozhraní API, které zaručuje podporu v novějších verzích této platformy [30]:

- **libc headers** – C knihovna
- **libm headers** – matematická knihovna
- **libz headers** – Zlib komprese
- **liblog header** – logování
- **libjnigraphics header** – přímý přístup do picel bufferu
- **C++ headers** – základní knihovny C++
- **OpenGL ES** – 3D grafika a audio

- **JNI rozhraní**

Nativní kódy jsou v projektu uloženy do složky *jni*. K překladu slouží skript *ndk-build* z balíčku NDK, který po spuštění provede kompilaci zdrojových a hlavičkových C/C++ souborů. Jsou vytvořeny nové soubory *obj* (přeložené objektové soubory) a *libs* (výsledné nativní knihovny .so). V Javě se k funkcím v nativním kódu dostaneme pomocí rozhraní JNI (Java Native Interface). Je potřeba načíst vytvořenou knihovnu a definovat metody, které budou v C/C++ implementovány. Potom je potřeba ve složce *jni* vytvořit soubor *jni.h*, který lze pomocí nástroje javah a již definovaných nativních funkcí v Java souboru vygenerovat. Tento soubor obsahuje hlavičky funkcí v nativním kódu a slouží k propojení s Java kódem. Navíc je potřeba ještě vytvořit soubory (opět složka jni) *Application.mk* a *Android.mk*. První soubor popisuje, které nativní moduly jsou aplikací požadovány, a také definuje některé globální proměnné. Druhý soubor je opět makefile soubor pro překladový systém NDK. Seskupuje zdrojové C/C++ soubory do modulů, které pak tvoří sdílené nebo statické knihovny. Hlavičkové soubory jsou systémem zjištěny automaticky, proto je není potřeba již uvádět [18].

7 Návrh aplikace

V rámci této práce byla navržena a implementována aplikace, která zatím slouží pro otestování použitých technologií, avšak do budoucna je určena pro prodej na Android marketu. V tomto obchodě již existuje hodně slovníkových aplikací, ale málokterá má takou slovní zásobu jako slovníky společnosti Lingea. Navíc díky převodu textu na řeč, tak dostaneme mnohem více než pouhý slovník. Aplikace se stane uživatelsky přívětivější a umožní mnohem více funkcí, než nabízí konkurence. Pomocí rozkladu fonémů jednotlivých slov si bude moc každý uživatel srovnat svoji výslovnost s rodilým mluvčím a zároveň si ji i vylepšit. V této kapitole jsou popsány specifikace požadavků na aplikaci a její návrh se strukturou a síťovou komunikací.

7.1 Specifikace požadavků

V prvním kroku bylo za úkol vytvořit mobilní aplikaci, která bude sloužit jako slovník a bude konkurovat ostatním aplikacím na trhu. Zvolená platforma byla Android, tedy jazyk Java kombinovaný s nativním kódem v C/C++. Tento slovník bude využívat již napsané knihovny (od Lingei) pro práci s uloženými daty (slovníky Lingea) a zároveň se stane plnohodnotnou aplikací určenou na prodej. Ve druhém kroku se měly integrovat technologie od společnosti Phonexia, což zahrnuje rozpoznávač řeči a rozdělení slov na jednotlivé fonémy. Tyto části nejsou součástí mobilní aplikace, ale poběží na vzdáleném serveru a slovníky se k ní budou připojovat. Hlavními požadavky na aplikaci byly:

- Vytvořit univerzální aplikaci, která nebude závislá na datech a bude sloužit jako základ pro všechny slovníky.
- Musí fungovat jak na mobilech, tak na tabletech.
- Jednoduché ovládaní na mobilních přístrojích.
- Možnost mít více slovníků v jedné aplikaci.
- Aplikace musí být chráněna proti neoprávněnému kopírování a používání.
- Servery musí obsluhovat více klientů zároveň.

Toto je výčet pouze takových hlavních bodů zadání. Celá specifikace je ovšem mnohem podrobnější a zabývá se rozvrženým jednotlivých grafických prvků, používanými gesty na displeji, návrhem komunikačních protokolů, šifrováním atd. Některé tyto věci budou popsány v následujících kapitolách.

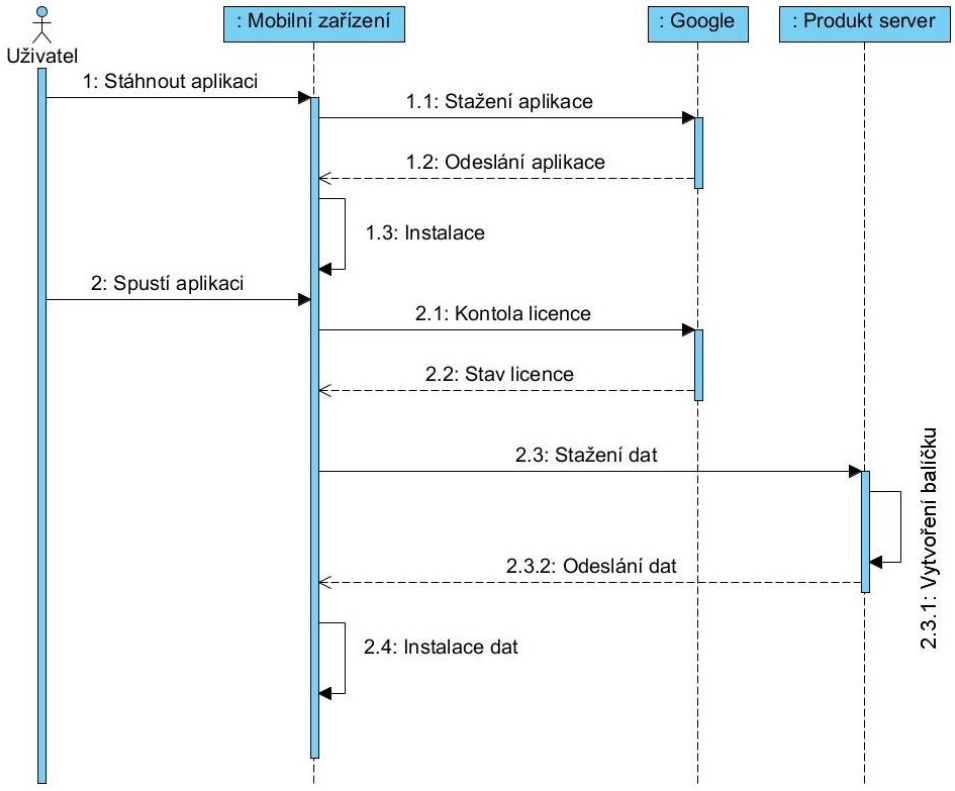
7.2 Struktura aplikace

Celková funkční aplikace se neskládá pouze z jedné části, ale hned z několika, které mezi sebou komunikují a spolupracují. Můžeme je rozdělit do čtyř takových hlavních částí:

- **Android aplikace** – Běží na mobilním zařízení a je to hlavní část celé aplikace. Můžeme ji opět rozdělit na dvě části:
    - **Java** – V jazyku Java je napsáno celé uživatelské rozhraní a také obsluha různých HW prvků (displej, mikrofon, reproduktor, …)
    - **C/C++** – Obsahuje jádro celého slovníku. Vyhledává požadovaná slova a vrací výsledky (synonyma fulltexty, …)
- **Server pro stahování dat** – Android aplikace si z něho stahuje potřebné slovníky. Vytváří tedy balíčky, které podepisuje, šifruje a posílá zpět. Tento server běží pod systémem Windows a je napsán v jazyce C#. Komunikace s ním probíhá pomocí POSTu a šifrovaných zpráv.
- **Server pro rozpoznávání** – Tady běží rozpoznávač řeči a také hledač fonémů. Je napsán v jazyce C++ pro platformu Linux. Komunikace probíhá pomocí spolehlivého protokolu TCP.
- **Google server** – Přes něj si uživatelé mohou stáhnout aplikaci. Navíc slouží jako zprostředkovatel plateb při koupi placeného obsahu a také ověřuje pravost aplikací.

7.2.1 Instalace aplikace

Obrázek níže znázorňuje první kroky pro získání a zprovoznění Android aplikace. Je na něm zobrazena komunikace mezi zúčastněnými stranami, která vede k úspěšnému nainstalování programu. Neúspěšné stavy zde nejsou zachyceny.



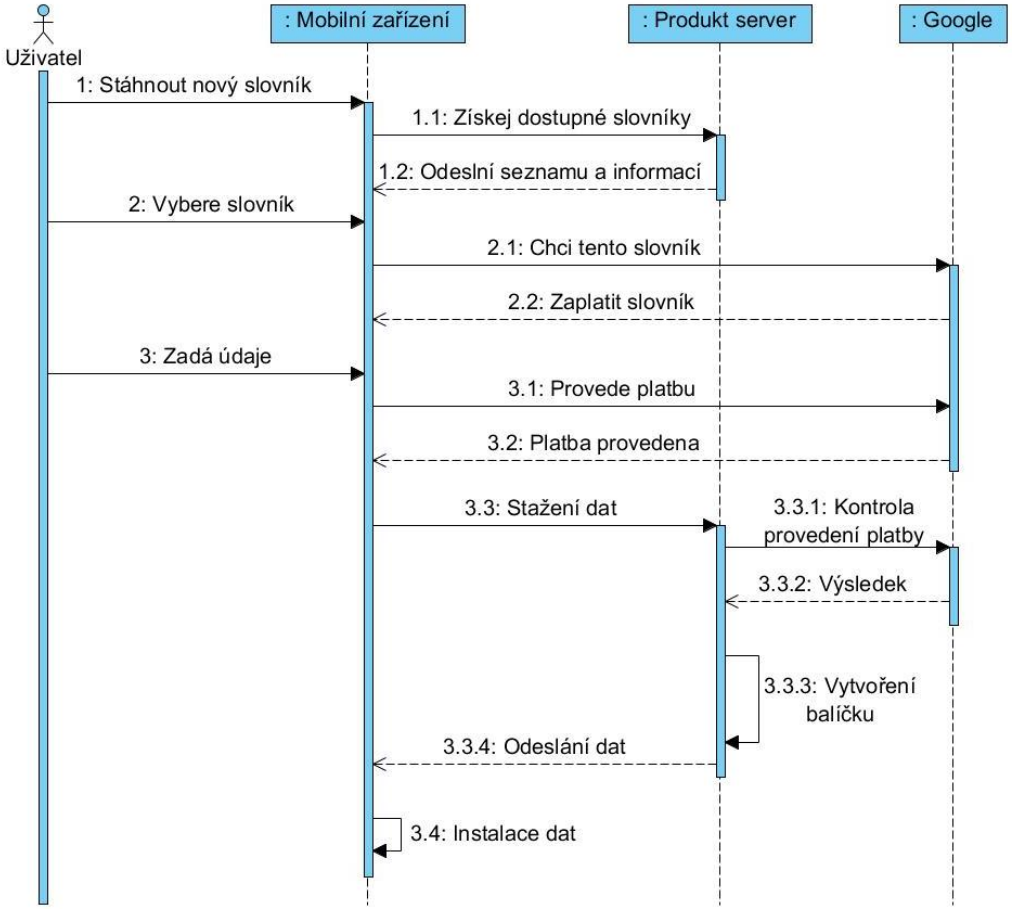
Obrázek 14: Instalace aplikace

Jak bylo uvedeno ve specifikaci, je potřeba vytvořit mobilní aplikaci nezávislou na datech. Proto samotná aplikace, stažená z Android Marketu, nemá žádnou velkou funkčnost. Aby se s této aplikace stal plnohodnotný slovník, tak je potřeba do něj ještě po instalaci stáhnout požadovaná data. Celý postup instalace je tedy následující:

1. Stáhnout aplikaci z Android Marketu.
2. Instalace na mobilním zařízení.
3. Při prvním spuštění je přes Google ověřena totožnost uživatele. Aplikace musí mít tedy možnost připojit se poprvé k internetu. Bez tohoto ověření nám slovník nepojede. Tento krok je potom prováděn vždy, když je aplikace spuštěna a má přístup k internetu.
4. Aplikace požádá server pro stahování dat o konkrétní slovník (dle verze aplikace).
5. Server vytvoří balíček, podepíše ho, zakóduje a odešle zpět.
6. Aplikace si rozbalí data a může je začít používat.

7.2.2 Instalace nových slovníků

Tento systém se anglicky nazývá In-App Billing a jedná se o koupi zpoplatněného obsahu přímo v aplikaci. Uživatel nepotřebuje nic hledat, ale stačí mu jeho základní aplikace, na kterou si může koupit další slovníky.



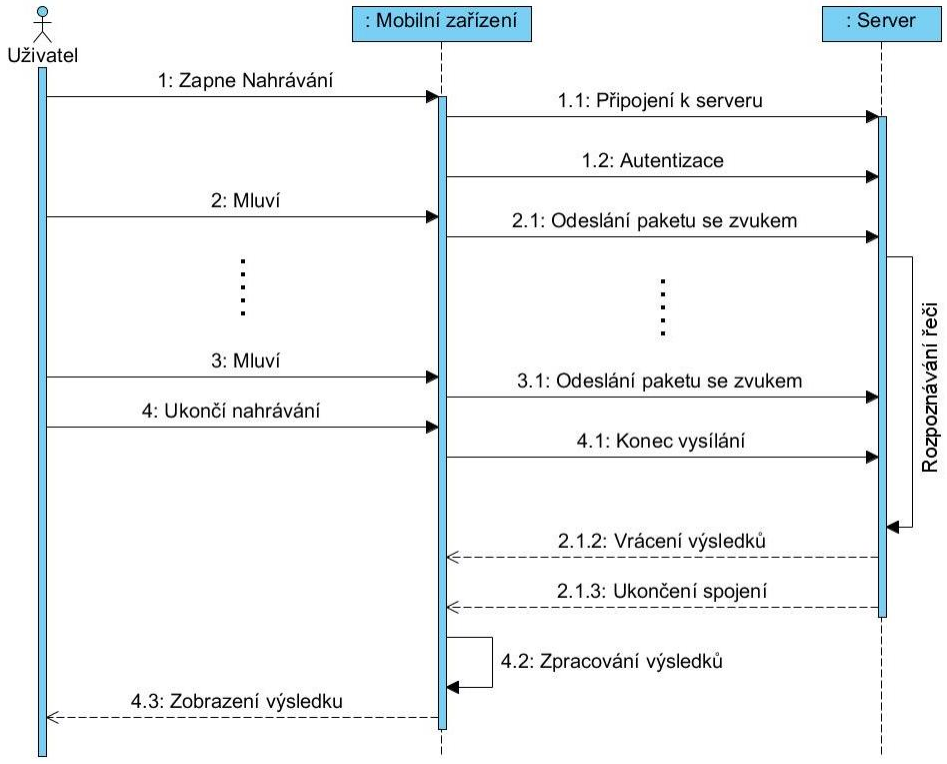
Obrázek 15: Stažení nových slovníků

Nemusíme mít tak ve svém zařízení několik samostatných slovníků, ale bude stačit pouze jeden, ve kterém si budeme moci přepínat mezi jednotlivými jazyky. Platba probíhá pomocí bankovního příkazu, který zprostředkovává Google. Při reinstalaci aplikace nemusíme mít ani strach, že o data přijdeme, protože záznamy si Google uchovává a aplikace kromě základních dat tak stáhne i ta dokoupená. Příklad komunikace aplikace se servery:

1. Aplikace se vyžádá seznam dostupných slovníků, které jsou k dispozici ke koupi.
2. Server s daty jí tento seznam zašle. Zasílají se i informace o daném balíčku (cena, popis).
3. Uživatel si ze seznamu vybere slovník, který by se mu hodil, a na Google se odešle id požadovaného balíčku.
4. Provede se platba za požadovaný obsah.
5. Aplikace opět osloví server s daty a požaduje od něj koupený slovník.
6. Server si z Googlu ověří, zda platba proběhla a v pozitivním případě vytvoří balíček, který odešle zpět.
7. Aplikace si data stáhne a připojí k již existujícím slovníkům.

7.2.3 Zpracování řeči

Mobilní zařízení komunikuje se serverem pomocí protokolu TCP, který nám zaručí spolehlivý přenos dat. Na straně klienta se tak musíme starat pouze o záznam zvuku, jeho překódování a odeslání serveru. Server si při připojení klienta vytvoří nové vlákno, na kterém probíhá příjem dat, dekódování a odeslání rozpoznávači.



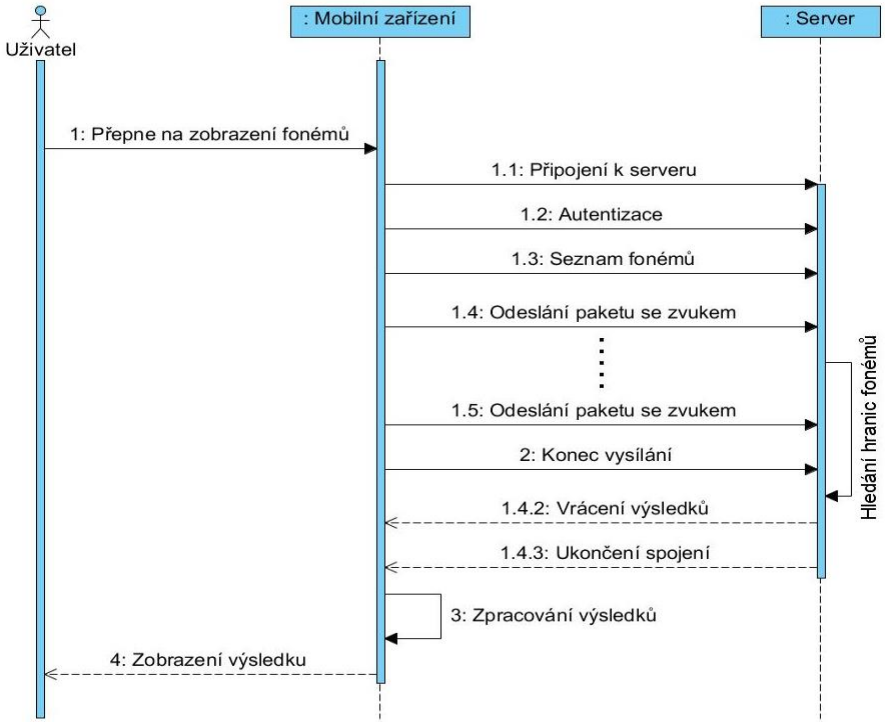
Obrázek 16: Rozpoznávání zvuku

Aby byla rychlost rozpoznávání co nejvyšší, tak se zvuk na server posílá po částech už ve fázi jeho nahrávání a server posílá jednotlivé části rozpoznávači, který už provádí převod zvuku na text. Nemusíme tak čekat na straně klienta na získání celého zvuku, potom na jeho přenos a nakonec rozpoznání, ale provádíme vše v „reálném čase“. Internet v mobilu ještě není zcela rozšířený a tomu odpovídají i sazby operátorů či limit přenesených dat, proto je potřeba brát ohled ještě na velikosti jednotlivých paketů nebo dobu připojení. Každý zvukový paket je proto ještě zakódován, aby byla zmenšena jeho velikost, a doba připojení je minimalizována na minimum. Posloupnost jednotlivých kroků je následující:

1. Uživatel zapne nahrávání.
2. Zařízení se připojí k serveru, proběhne jeho autentizace a přenos parametrů (co má sever dělat, zvolený kodek, požadovaný výstup).
3. Mobilní zařízení po jednotlivých časových úsecích získá nahraný zvuk, zkomprimuje ho a odešle na server.
4. Server přijatý paket dekóduje a pošle rozpoznávači.
5. Kroky 3 a 4 se opakují, dokud uživatel nahrávání nezruší nebo neuplyne maximální povolený limit nahrávání.
6. Server odešle získané výsledky a ukončí spojení.

7.2.4 Hledání hranic fonémů

Poslední částí je získání hranic fonémů. Server kromě zvuku potřebuje ještě seznam fonémů daného hesla, který musí získat ještě před přenosem zvukové stopy. Jinak komunikace probíhá podobně jako je tomu u rozpoznávání zvuku. Na obrázku je znázorněno odeslání již nahraného zvuku od rodilého mluvčího. Jinak je možné nahrát a odeslat zvuk vlastní, to je potom podobné předešlému diagramu.



Obrázek 17: Hledání hranic fonémů

8 Implementace klientské části

Tato kapitola se zabývá popisem použitých technik a implementace klientské části aplikace. Jak už bylo zmíněno v předchozích kapitolách, aplikace bude implementována pro mobilní zařízení se systémy Android a bude využívat co možná nejnižší potřebné API. Díky tomu ji budou moci používat i uživatelé se starším zařízením (využívající starší verzi Androidu).

8.1 PCM

Pulzně kódová modulace (PCM) [32] je metoda převodu analogového zvukového signálu na signál digitální. Tato metoda je využita na mobilním zařízení s Androidem. Používá vzorkovací frekvence (s rozlišením 16 bitů na vzorek):

- 8 kHz pro rozpoznávání řeči
- 16 kHz pro nalezení hranic fonémů

Tento formát se ale pro přenos nehodí, neboť zabírá mnoho místa. Když si vezmeme rozpoznávání řeči s 8 kHz, 16 bitů na vzorek, mono a velikost slova 2 sekundy, tak nám vyjde velikost záznamu $\frac{8000*16*2}{8} = 32000$ bytů. Pro zmenšení velikosti zakóduji zvuk do speexu (ztrátová komprese), který ho dokáže zmenšit až na desetinu jeho původní velikosti (záleží na nastavení požadované kvality). Na serveru je potom potřeba speex opět dekódovat na PCM.

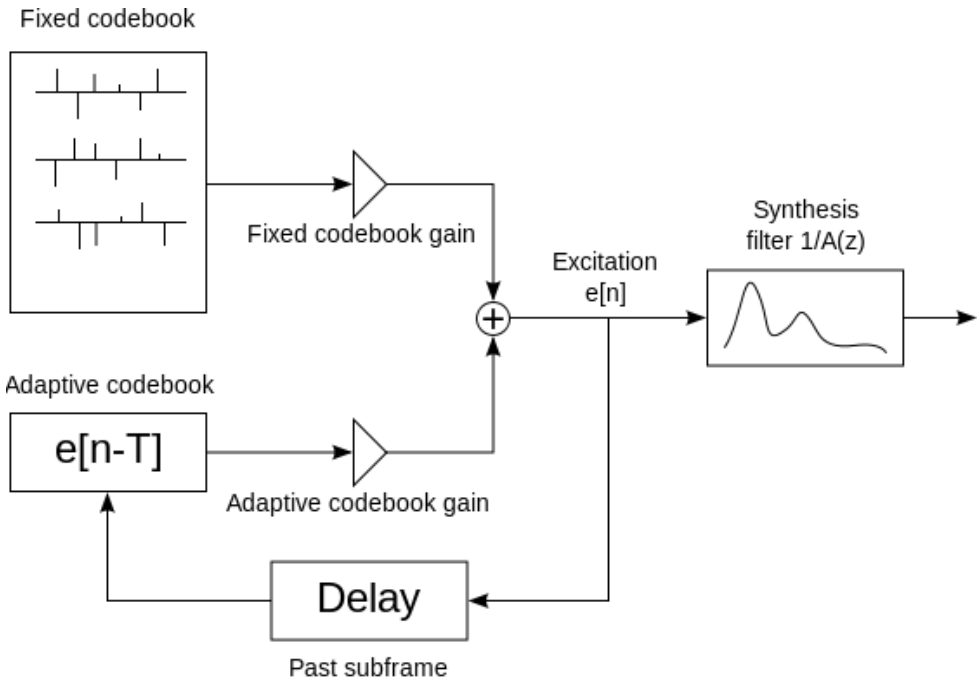
8.2 SPEEX

Speex je svobodný formát (Open Source) zvukové komprese navržený a optimalizovaný pro lidskou řeč. Je součástí ogg projektu. Speex je vhodný k VoIP komunikaci, audio streamingu, k archivaci (hlasová pošta) a slouží také jako formát zvukových knih. Tento kodek je založen na CELP (Code excited linear prediction) algoritmu a je určen pro kompresy hlasu v rozmezí 2 až 44 kbps. Speex byl navržen pro použití se třemi vzorkovacími frekvencemi 8 kHz, 16 kHz, a 32 kHz (narrowband, wideband a ultra-wideband). Kvalita se nastavuje jako parametr od 0 do 10 [33].

8.2.1.1 CELP

Tato technika je založena na třech myšlenkách [34]:

1. Využití modelu lineární predikce k modelování zvukového ústrojí
2. Využít (adaptivní a pevnou) kódovou knihu jako vstup (buzení) LP modelu
3. Vyhledávání se zpětnou vazbou



Obrázek 18: CELP model [35]

8.2.1.2 Úzkopásmový (narrowband) a širokopásmový (wideband) mód

Úzkopásmový mód pracuje se vzorkovací frekvencí 8 kHz, velikost okna je 20 ms (což odpovídá 160 vzorkům). Každé okno je ještě rozděleno na 4 další okna po 40 vzorcích. Tento mód byl použit pro kódování přenášené zprávy při rozpoznávání slov [36].

Pro širokopásmový mód, Speex využívá QMF (Quadrature mirror filter), které rozdělí pásmo na dvě části. 16 kHz signál je tedy rozdělen na dva 8 kHz signály, kde jeden představuje dolní pásmo (0-4 kHz) a druhý vysoké pásmo (4-8 kHz). Dolní pásmo je kódováno, jako by to byl úzkopásmový signál a druhé pásmo je kódováno zvlášť. Tento mód byl použit pro kódování a dekódování zvuku při zjišťování jednotlivých hranic fonémů [37].

Mód	Kvalita	Bitrate (bps)	Kvalita / popis
0	-	250	No transmission (DTX)
1	0	2,150	Vocoder (mostly for comfort noise)
2	2	5,950	Very noticeable artifacts/noise, good intelligibility
3	3-4	8,000	Artifacts/noise sometimes noticeable
4	5-6	11,000	Artifacts usually noticeable only with headphones
5	7-8	15,000	Need good headphones to tell the difference
6	9	18,200	Hard to tell the difference even with good headphones
7	10	24,600	Completely transparent for voice, good quality music
8	1	3,950	Very noticeable artifacts/noise, good intelligibility

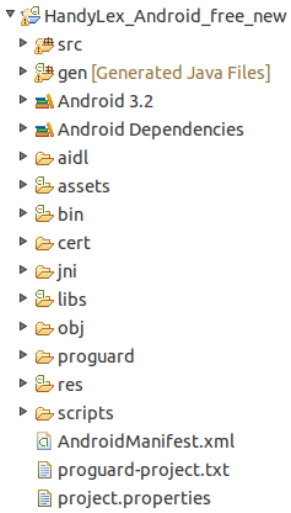
Tabulka 1: Kvalita speexu (úzkopásmový mód) [36]

Kvalita	Bitrate (bps)	Kvalita / popis
0	3,950	Barely intelligible (mostly for comfort noise)
1	5,750	Very noticeable artifacts/noise, poor intelligibility
2	7,750	Very noticeable artifacts/noise, good intelligibility
3	9,800	Artifacts/noise sometimes annoying
4	12,800	Artifacts/noise usually noticeable
5	16,800	Artifacts/noise sometimes noticeable
6	20,600	Need good headphones to tell the difference
7	23,800	Need good headphones to tell the difference
8	27,800	Hard to tell the difference even with good headphones
9	34,400	Hard to tell the difference even with good headphones
10	42,400	Completely transparent for voice, good quality music

Tabulka 2: Kvalita speexu (širokopásmový mód) [37]

8.3 Struktura nového projektu

Při vývoji aplikace pro platformu Android je potřeba zachovat určitou strukturu projektu, se kterou vývojové nástroje pracují. Na následujícím obrázku je znázorněno základní rozložení:



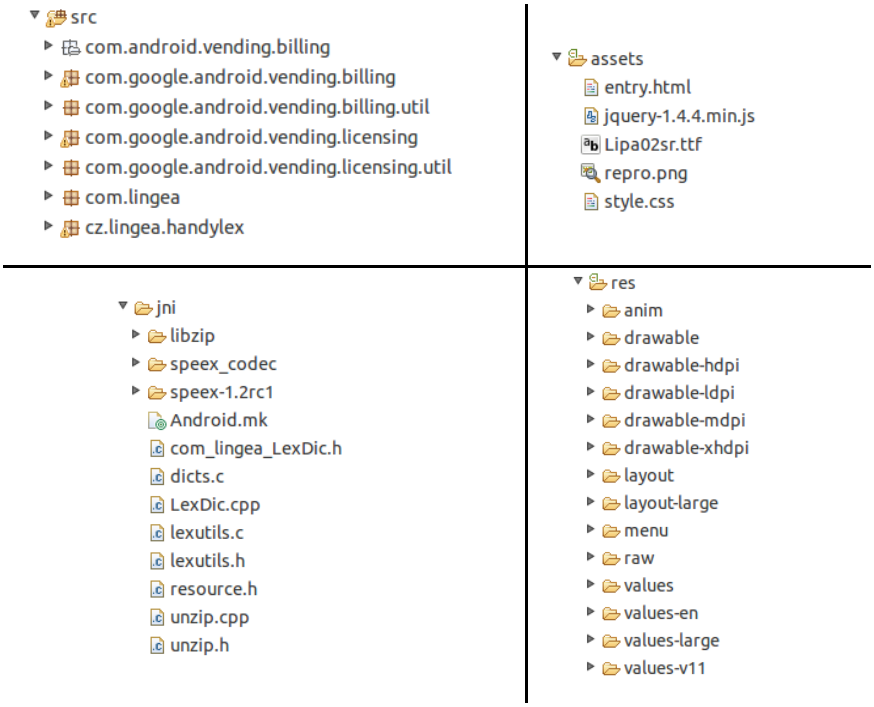
Obrázek 19: Struktura založeného projektu

Každá aplikace musí obsahovat soubor **AndroidManifest.xml** (s právě tímto názvem) v kořenovém adresáři. Manifest prezentuje základní informace o aplikaci v systému Android. Tyto informace systém potřebuje mít ještě před spuštěním kódu aplikace. V tomto souboru je potřeba uvést následující informace [38]:

- Jméno Java balíčku pro aplikaci. Název balíčku slouží jako jedinečný identifikátor pro danou aplikaci.
- Popisuje komponenty aplikace (aktivity, služby, přijímače vysílání, a dodavatele obsahu), ze kterých je aplikace tvořena. Jmenuje názvy tříd, které implementují každou komponentu a uvádí její možnosti (například, které zprávy – *intenty* mohou přijímat). Tato deklarace dává vědět systému Android, jaké komponenty má a za jakých podmínek mohou být spuštěny.

- Dále uvádí, která oprávnění aplikace musí mít, aby mohla přistupovat ke chráněným částem API a komunikovat s jinými aplikacemi nebo jaká oprávnění potřebují jiné aplikace ke komunikaci s ní.
- Udává minimální verzi použitého API.
- Seznam knihoven, které aplikace používá.

Soubory **proguard-project.txt** a **project.propreties** slouží ke zmenšení, optimalizaci a obfluskaci kódu. ProGuard [39] nástroje odstraňují přebytečný kód a přejmenovávají třídy, pole a metody významově nesmyslnými jmény. Výsledkem je menší velikost *.apk* souboru, který je těžší zpětně analyzovat. Jedná se o jednu z možností jak zabezpečit aplikaci, která je použita i v této aplikaci. Navíc je tento nástroj integrován do překladače, a proto je jeho použití docela jednoduché.



Obrázek 20: Struktura důležitých složek projektu

Nebudou zde popsány všechny části této adresářové struktury, ale pouze ty, jenž byly pro projekt významné. Na obrázku 20 jsou znázorněny čtyři složky, které je třeba si alespoň trochu vysvětlit. Jedná se konkrétně o složku *src*, ve které jsou uloženy všechny Java soubory rozdělené do jednotlivých balíčků. Balíčky od Googlu obsahují třídy potřebné k nákupu placeného obsahu a ověření licence aplikace. V *com.lingea* jsou třídy, které slouží k propojení s nativním kódem C/C++ (popsáno v kapitole o NDK) a balíček *cz.lingea.handylex* obsahuje třídy tvořící aplikaci.

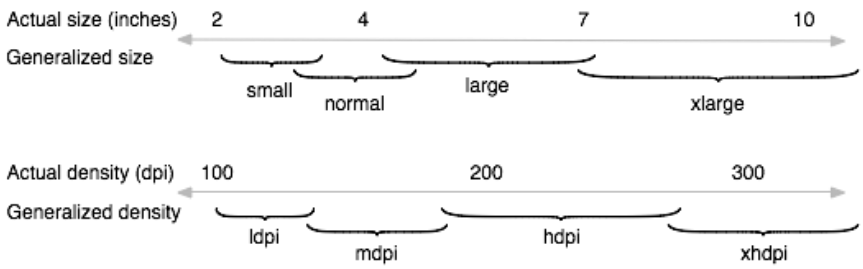
Ve složce *jni* nalezneme zdrojové kódy psané v jazyce C/C++ (a další potřebné pro překlad, viz kapitola o NDK). Jsou zde zdrojové kódy pro kódování do speexu, rozbalování stažených balíčků a taky jádro aplikace (dodané Lingeou) sloužící pro práci se staženými daty (vyhledávání hesel, zobrazování nastavení, fulltextů atd.).

Složka *assets* bývá určena pro dodatečné soubory aplikace a je ve většině případů prázdná. Jelikož jsou data slovníků ve formátu HTML, tak bylo potřeba vytvořit stánku, kde se budou zobrazovat a nadefinovat jednotlivé styly a akce. Proto složka *assets* obsahuje tyto soubory.

A nakonec je zde složka *res*, která slouží pro definici grafiky, textů, animací apod. U Androidu máme možnost psát grafiku přímo v kódu aplikace nebo můžeme vytvářet XML dokumenty popisující jednotlivé obrazovky a jejich prvky. My jim potom v aplikaci dopíšeme pouze funkčnost. ADT plugin obsahuje prostředí, ve kterém je grafický editor GUI. V souboru *res* se mohou objevit tyto složky:

- **anim** – Obsahuje soubory popisující animaci. Většinou se jedná o posuvy, rotace a zprůhlednění v nějakém časovém rozpětí.
- **drawable** – Tady jsou uloženy všechny soubory popisující jednotlivé prvky GUI (rámy, ikony, definice vlastních stylů, obrázky, bitmapy, …).
- **layout** – V této složce se nachází soubory s grafickým popisem jednotlivých obrazovek (aktivit).
- **menu** – Popisuje vzhled systémového menu (to co se zobrazí v aplikaci, klikneme-li na tlačítko menu).
- **values** – Obsahuje texty a popisky aplikace.

Navíc složky *drawable*, *layout* a *menu* se mohou vytvořit pro různé rozlišení obrazovek. To se udělá velice jednoduše. Stačí vytvořit stejně pojmenovaný adresář a ke jménu připojit typ obrazovky (obrázek 20). Jestliže tyto složky obsahují stejně pojmenované soubory, tak si aplikace vybere ten, který bude v odpovídající složce. K *values* se navíc může přidat přípona určující jazyk popisků a dle nastaveného jazyku telefonu se vybere i složka *values*. Tento způsob velice ulehčil práci při psaní GUI pro mobilní zařízení a potom zvlášť pro tablety.



Obrázek 21: Volba rozlišení [40]

8.4 Aplikace pro mobilní telefon

Poskytovatelé obsahu

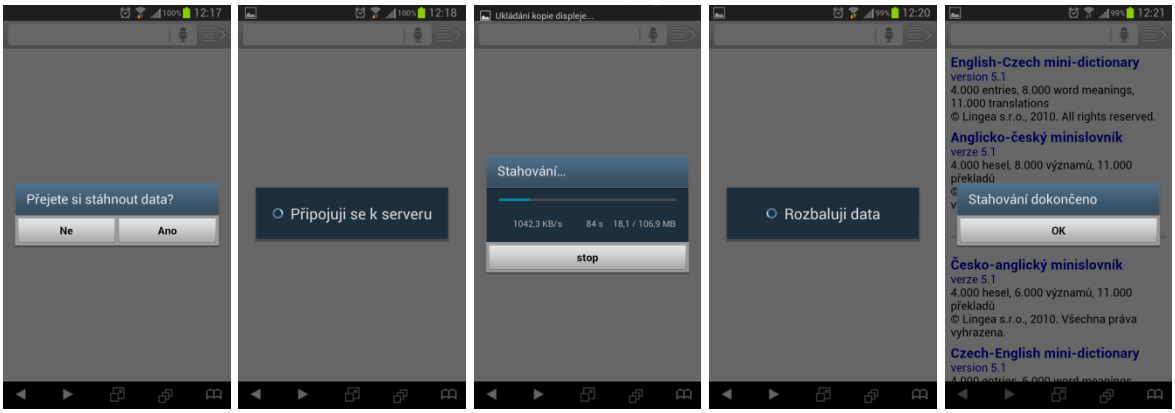
Jak už bylo zmíněno v kapitole 6.3.2, Android využívává několik způsobů pro práci s daty. V této aplikaci se data ukládají standardně na paměťovou kartu (nebo přímo do paměti telefonu) do složky *Android/data/*. Tato složka u většiny android aplikací slouží právě k tomuto účelu. Jsou zde stažené a podepsané slovníky, ze kterých aplikace potom vyhledává. Navíc velké slovníky obsahují namluvené hesla (od rodilých mluvčích), která jsou zde uložena taky (samozřejmě ve zkomprimované podobě).

Dále používá Shared Preferencies pro uložení uživatelského nastavení. Toto se hodí, protože data zůstanou uložena i po vypnutí aplikace a většina nastavení odpovídá potřebným podmínkám pro tuto metodu ukládání. Dalo by to přirovnat jako ukládání pomocí cookies.

Asynchronní úlohy

Každá aplikace v Androidu má vlastní UI vlákno. V tomto vláknu běží všechny komponenty aplikace (aktivity, služby, poskytovatelé obsahu a příjemce broadcastu). Systém Android nedovoluje komponentám blokovat vlákno s uživatelským rozhraním déle než několik sekund (kolem pěti). Jakákoliv delší činnost pak může způsobit pád aplikace. Proto by se všechny déle trvající operace měli implementovat ve vláknech na pozadí. Od verze Androidu 1.5 (API 3) je k dispozici třída **AsyncTask**, která je právě takovou koncepcí pro práci s vlákny na pozadí. Zahrnuje v sobě veškerou potřebnou funkcionalitu pro práci s vlákny a komunikací s UI vláknem. Android si tato vlákna sám alokuje a po skončení je taky odstraní. Třída *AsyncTask* obsahuje několik důležitých metod [41]:

- **onPrepareProgress()** – Volá se před spuštěním vlákna na pozadí. Může se vyvolat dialog V UI vláknu či provést nastavení běhu nového vlákna.
- **doInBackground()** – Uvnitř této metody se definuje, co přesně se má provést na pozadí. Tato metoda je jediná povinná.
- **onProgressUpdate()** – V tělo této metody je kód který se provede po zavolání funkce *onPublishProgress()*, kterou je možné zavolat v *doInBackground()*. Jedná se vlastně o interakci s vláknem UI během provádění nějaké akce na pozadí (progres stahování, hlasitost nahrávaného zvuku, …).
- **onPostExecute()** – Zavolána po skončení metody *doInBackground()*. V aplikaci slouží pro zpracování získaných dat.
- **onCancelled()** – Obdoba předchozí případu, jenom se zavolá, pokud bylo vlákno v průběhu jeho činnosti ukončeno.

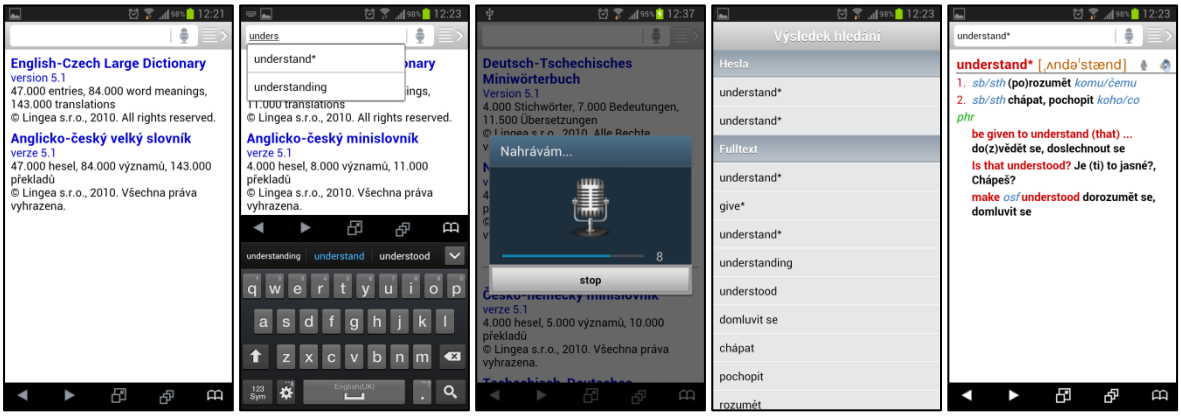


Obrázek 22: Posloupnost při instalaci dat

Na obrázku 22 vidíte posloupnost jednotlivých kroků při prvním spuštění aplikace a stažením potřebných slovníků. Jako první se aplikace zeptá, zda máte zájem si data stáhnout. Tento krok je

nezbytný, protože u větších slovníků (což může být až 100 MB) by si mohl uživatel chtít změnit způsob připojení a stáhnou je později. Při kladné odpovědi je potom zahájena komunikace se serverem (pomocí *AsyncTask*) a jsou uživateli odeslány vytvořené balíčky. Zařízení data stáhne, rozbalí a aplikace je připravena k použití. Jestliže data nestáhne, tak se zobrazí pouze prázdné okno a po každém novém spuštění se aplikace zeptá, zda si přejeme data stáhnout.

Obrázek 23 a) ukazuje úvodní stránku a zároveň i rozložení celé obrazovky v hlavní aktivitě. Nahoře je lišta, ve které máme zadávací pole s tlačítkem v podobě mikrofonu a ještě tlačítko pro přepnutí na obrazovku s výsledky hledání (obrázek 23 d). Uprostřed se nachází prvek nazývaný *WebWiew* a vyplňuje skoro celou obrazovku. Slouží pro zobrazení webového obsahu. Právě do něj se vykreslují nalezené výsledky a jsou dále upraveny pomocí souborů ze složky *assets* (css, javascript). Spodní lišta má tlačítka pro pohyb ve slovníku a pro zobrazení dalších obrazovek s příklady použití daného slova či odkazy na další podobná hesla ze slovníku a zobrazení historie vyhledávání.



Obrázek 23: Vyhledávání ve slovníku: a) Úvodní obrazovka, b) Vyhledávání, c) Nahrávání zvuku, d) Výsledky hledání, e) Zobrazené heslo

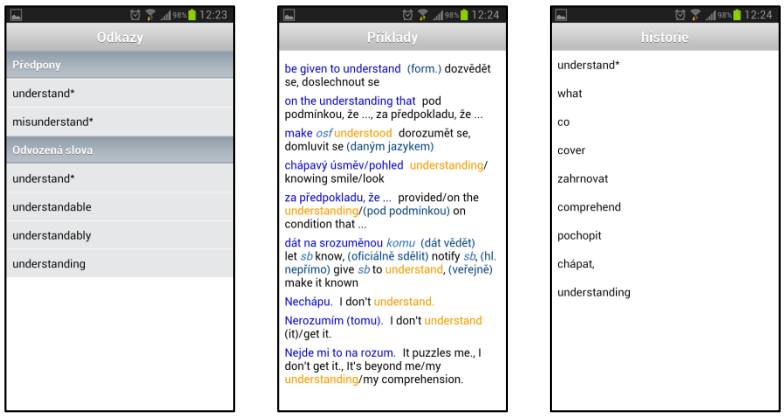
Zadávací pole je typu *AutoCompleteTextView* a dokáže nabízet možnosti hledaného textu dle již napsaných znaků (obrázek 23 b). Mikrofon v tomto poli spouští funkci převodu řeči na text. V telefonu se tak vytvoří připojení k rozpoznávacímu serveru a po autentizaci přístroje probíhá nahrávání a přenos dat (obrázek 23 c). K tomu opět využijeme třídu *AsyncTask*, která nám vytvoří vlákno na pozadí. V tomto vláknu probíhá právě zmíněné připojení k serveru, autentizace, nahrávání zvuku, jeho fragmentace a kódování do speexu (v nativním kódu), odesílaní na server a nakonec příjem zpracovaných informací. V průběhu tohoto procesu nám v UI vláknu běží časový odpočet (nastavený na 10 vteřin) a zobrazuje se hlasitost nahrávaného zvuku. Po skončení komunikace nám aplikace nabídne získané výsledky jako možnosti v zadávacím poli.

Výsledky vyhledávání (obrázek 23 d) jsou zobrazeny v nové aktivitě jako *ListView*. Tato komponenta nám daný seznam nebo pole převede do grafické podoby a zobrazí na obrazovce. Na každý prvek je potom možné kliknout, čímž se tato aktivita ukončí a vrátí výsledek zpět do hlavní aktivity, která ho zobrazí (obrázek 23 e). Je zde použit ještě adaptér, který nám obarvuje jednotlivé prvky pole.

Hlavní aktivita (konkrétně prvek *WebView*) obsahuje třídu pojmenovanou *GestureListener*. Tato třída snímá pohyby gest v tomto prvku a při přejíždění zleva doprava a obráceně nám přepíná mezi slovy. Lze to používat místo spodních šipek k zjednodušení ovládání. Navíc každé slovo slouží zároveň jako odkaz do slovníku a při kliknutí na něj se provede jeho vyhledání. Vedle hesla je zobrazena i jeho výslovnost a dvě ikony. První je opět mikrofon a slouží pro nalezení hranic fonémů.

Druhá má tvar reproduktoru a ta přehrává uložený zvuk. Tyto ikony se zobrazují pouze u hesel, která mají nahraný příklad výslovnosti.

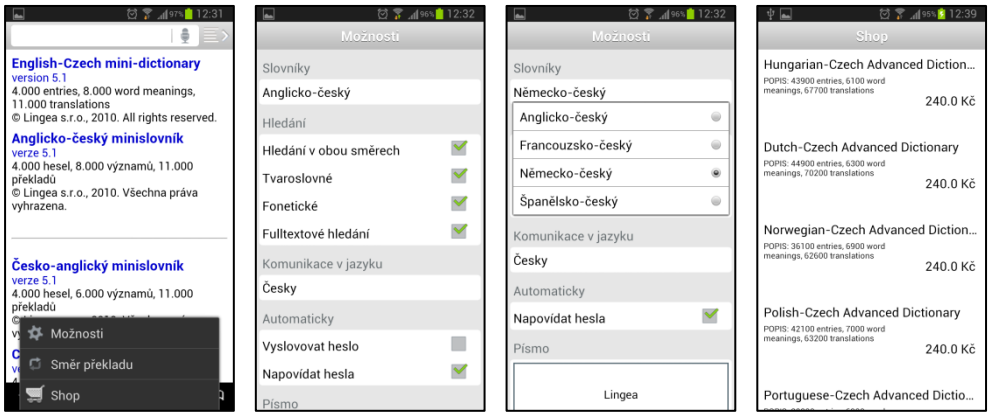
Aktivity s historií vyhledávání a odkazy na další slova jsou podobné aktivitě s výsledky hledání. Jedná se opět o *ListView*, ale je akorát naplněn jinými daty. Aktivita s příklady je tvořena převážně z *WebView*, kde jsou zobrazeny jednotlivé věty používající dané slovo. Každé slovo je opět odkazem do slovníku. Z aktivit je možné se vrátit tlačítkem back nebo si vybrat některé slovo a provést nové vyhledávání.



Obrázek 24: Další aktivity: a) Zobrazení odkazů, b) Zobrazení příkladů, c) Historie vyhledávání

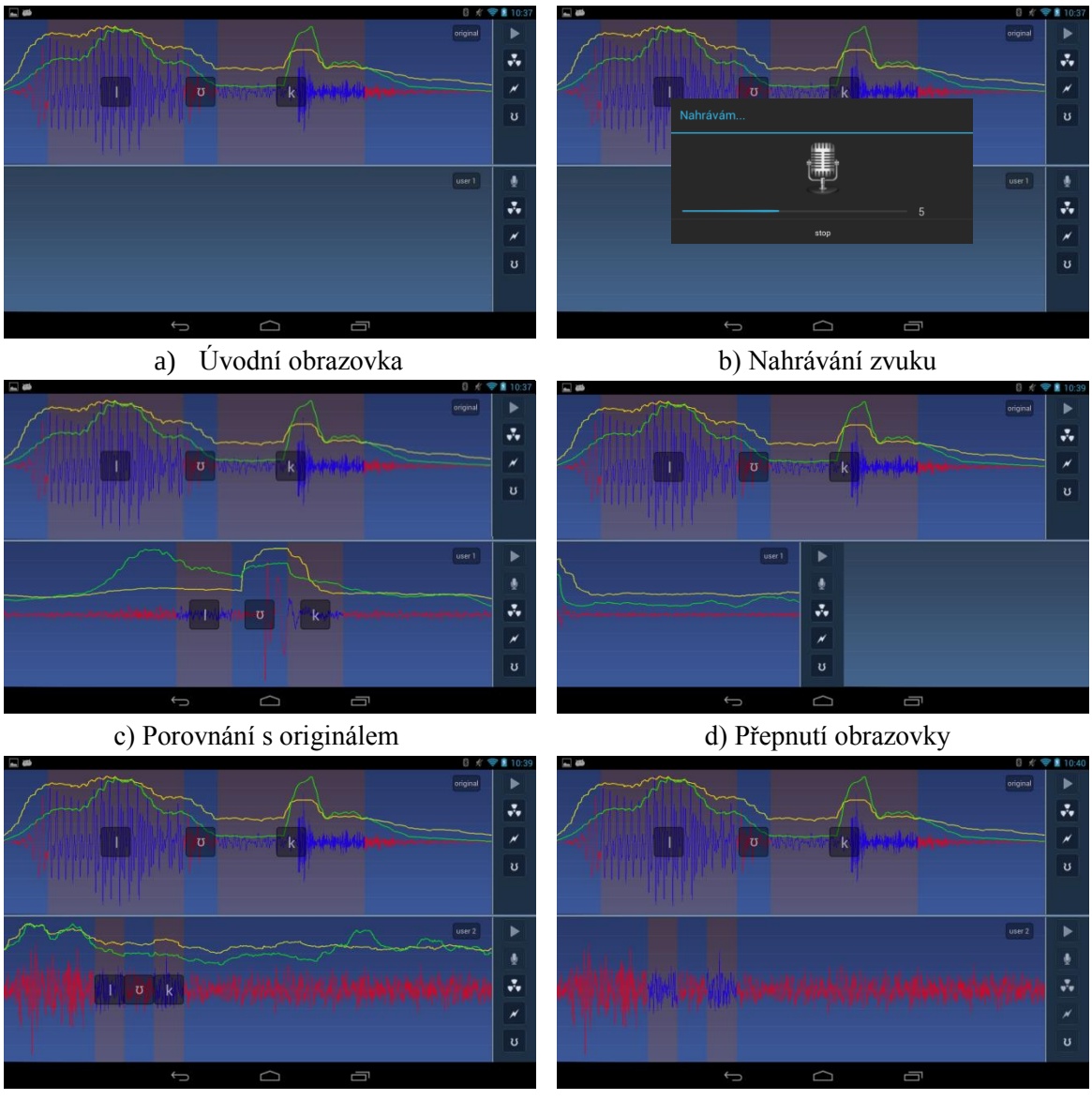
Ve složce *menu* tohoto projektu se nachází soubor popisující systémové menu (ikona a text). Toto menu se vyvolá stisknutím tlačítka menu, které má každý přístroj se systémem Android. Na obrázku 25 a) je zobrazeno a dovoluje nám vyvolat aktivitu s nastavením aplikace, otočit směr překladu (anglicko-český => česko-anglický) nebo zobrazit slovníky, které si lze dokoupit.

V možnostech si lze vybrat jazyk používaného slovníku, jazyk textů v aplikaci a nastavit různé možnosti vyhledávání. Obrázek 25 d) zobrazuje seznam dostupných slovníků možných k zakoupení. Komunikace opět probíhá přes třídu *AsyncTask*. Po zaplacení nákupu je opět spuštěno stahování (obrázek 22). Princip nákupu placeného obsahu je popsán již v kapitole 7.



Obrázek 25: Položky menu: a) Systémové munu, b) Možnosti aplikace, c) Změna slovníku, d) Nákup placeného obsahu

Posledním bodem implementace je vyhledávání hranic fonémů. Tento okamžik nastane, jestliže uživatel zmáčkne tlačítko mikrofonu zobrazené vedle vyhledaného slova. Spustí se nová aktivita, která je vodorovně rozdělena na dvě půlky. Každá půlka obsahuje komponentu zvanou *ViewPager*, která je naplněna polem obrazovek. Je to potom podobné úvodní obrazovce systému Android. Tažením obrazovky doleva nebo doprava se nám přepínají (obrázek 26 d). Pro všechny obrazovky slouží jako základ pouze jeden layout (obrázek 26 a – spodní půlka). Před odesláním zvuku na server (opět pomocí třídy *AsyncTask*) ještě musíme převést Lingea fonémy na AMI fonémy a poslat je jako první. Princip celé komunikace je popsán v kapitole 7. Lišta napravo obsahuje tlačítka pro interakci. Prvním je tlačítko play, které přehraje nahraný zvuk. Další tlačítko (s mikrofonem) slouží pro nahrávání zvuku (obrázek 26 b). Poslední tři tlačítka jsou pro zobrazení rychlosti, hlasitosti a fonémů (rozdíl mezi posledními dvěma obrazovkami z obrázku 26). Na vykreslení křivky jsem použil metodu *onDraw()*, ve které jsem kreslil pouze přímky (z bodu do bodu). Před samotným vykreslením bylo nutné si přepočítat jednotlivé souřadnice dle velikosti vykreslovací plochy (provedeno v C/C++).



a) Úvodní obrazovka

b) Nahrávání zvuku

c) Porovnání s originálem

d) Přepnutí obrazovky

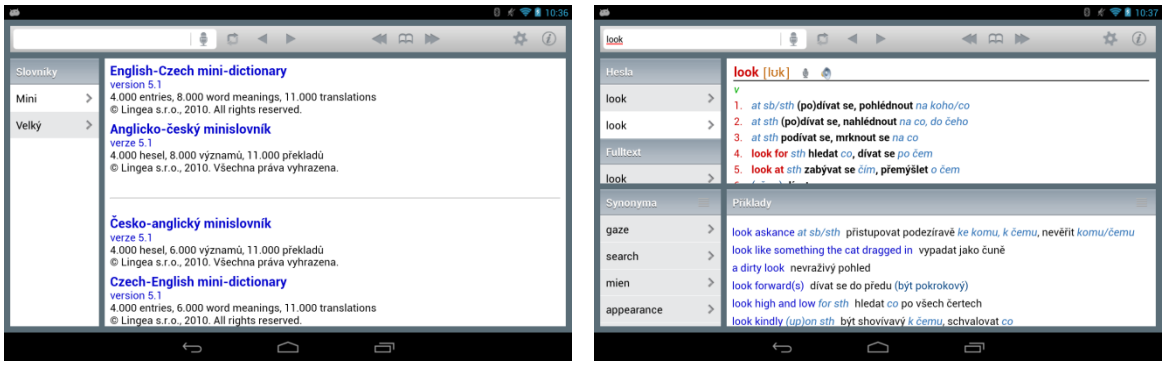
e) Zobrazené prvky

f) Skryté prvky

Obrázek 26: Zobrazení hranic fonémů

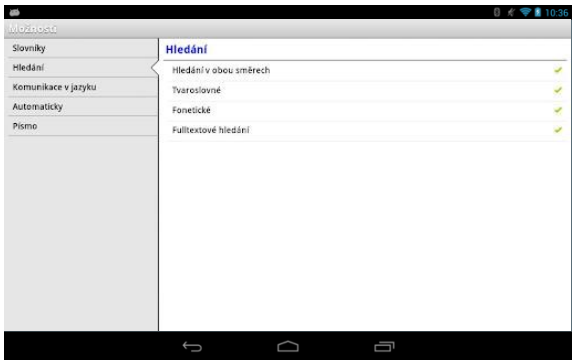
8.5 Aplikace pro tablet

Java kód zůstal celkem stejný jako je tomu u mobilní aplikace, nicméně bylo nutno přidat některé funkce pro obsluhu jiného grafického rozhraní. Stačilo pouze vytvořit nový layout, který se jmenuje stejně jako layout použitý na mobilu a umístit ho do složky *layout-large*. Díky tomuto kroku si zařízení s větším displejem zvolí tento soubor. Na obrázku 27 je vidět grafické zpracování hlavní aktivity. V prvním případě se jedná o úvodní obrazovku a ve druhé už o vyhledané heslo.



Obrázek 27: Hlavní aktivita pro tablet

Ve vrchní liště (navigační liště) se nachází pole pro vyhledávání, otočení slovníku, posun mezi vyhledávanými hesly a historií a tlačítko pro spuštění nové aktivity s nastavením. V levém sloupci jsou seznamy nalezených výsledků a jejich synonyma, fulltexty či překlepy. Jedná se o klasický *ListView*, ve kterém je použitý adaptér. Tento adaptér má za úkol probarvování jednotlivých prvků seznamu dle jejich obsahu. V pravém sloupci máme dvě okna, která jsou typu *WebView*, a slouží stejně jako je tomu na mobilu. Vrchní okno nám zobrazuje jednotlivá hesla a jejich význam. Ve spodním se nachází příklady použití hesel (pokud žádné nejsou, tak se vrchní okno roztáhne na celou obrazovku). Jinak je funkcionalita stejná, jako tomu bylo na mobilu.



Obrázek 28: Aktivita s nastavením pro tablet

Protože mají tablety větší displej tak nebylo nutné tolik aktivit (obrazovek) pro zachování stejné funkcionality aplikace. Stačily pouze tři. První byla popsána výše a druhá obsahuje nastavení aplikace a třetí rozklad fonémů. V nastavení je možné provádět stejné změny jako tomu je u telefonů, ale pro lepší orientaci a přehlednost bylo nutné přepracovat celý vzhled aktivity (obrázek 28).

9 Implementace serverové části

Druhou částí aplikace jsou i servery, se kterými aplikace komunikuje. Bez serveru na stahování produktů máme pouze holou aplikaci bez jakékoliv funkčnosti a druhý nám slouží k rozpoznávání řeči. V této kapitole se budeme zabývat tím druhým serverem.

9.1 BSAPI

Brno Speech Core (**BSCORE**) je sada základních komponent pro jednoduché a velmi rychlé rozpoznávání řeči. Implementuje širokou škálu algoritmů např.: čtení řeči ze souborů / mikrofonů, zpracovávání seznamu souborů, klasifikaci, dekódování, rozpoznávání fonémů, detekci klíčových slov, rozpoznávání jazyka a rozpoznávání řečníka. **Brno Speech Application Interface** (**BSAPI**) je rozhraní mezi BSCORE a ostatními aplikacemi.

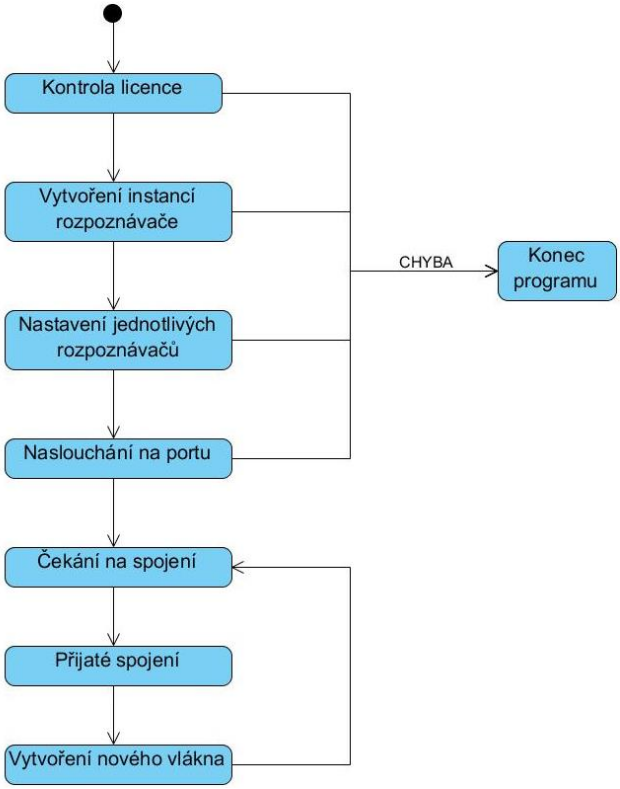
Aplikační rozhraní pro BSCORE je objektově orientované. Každý algoritmus má svoji třídu. Třídy jsou přístupné prostřednictvím rozhraní. Rozhraní je čistě virtuální třída (všechny komponenty jsou virtuální). Tyto třídy jsou na konci jména označeny písmenem I. Vývojář může vytvořit instanci pomocí funkce *BSAPICreateInstance()*. Všechna rozhraní jsou odvozena z třídy *SUnknownI*. Tím jsou zajištěny základní funkce, jako počítání referencí a odstranění instance z paměti, pokud ji nikdo nepoužívá [42].

9.2 Vlastní implementace

Server je implementován v jazyce C/C++ a běží na platformě Linux. Při psaní programu byly použity dvě externí knihovny. První z nich byla knihovna dovolující využívat BSAPI (pro převod řeči na text – zatím CZ a EN, pro rozpoznávání fonémů – pouze EN). Další knihovna slouží k převodu speexu na PCM (bylo experimentováno i s knihovnou opus). Spuštění programu a naslouchání na požadovaném portu zobrazuje obrázek 29. Stavy, kterých lze dosáhnou, jsou následující:

- **Kontrola licence** – V tomto bodě se kontroluje licence používání BSAPI. Jestliže se kontrola neprovede (nepřipojí se k serveru nebo licence vyprší), tak program skončí.
- **Vytvoření instancí rozpoznávače** – Vytvoření instance rozpoznávače řeči *SMulticoreOfflineSpeechRecognizer3I* a rozpoznávače fonémů *SNNetPosteriorEstimatorI*. V dřívějších verzích byl použit pouze *SOfflineSpeechRecognizer3I*, ke kterému musely být implementovány fronty na čekání. Tato nová verze již fronty obsahuje. Je vytvořeno více instancí pro paralelní (asynchronní) zpracování.
- **Nastavení jednotlivých rozpoznávačů –** Zde probíhá nastavení rozpoznávacích sítí, vytvoření obsluh chyb, obsluh jednotlivých instancí rozpoznávačů a jejich přiřazení příslušným instancím. Jestliže při nastavení dojde k chybě, tak program končí.
- **Naslouchání na portu** – V tomto kroku je nastaveno připojení a port, na kterém má program naslouchat. V případě nedostupného portu nebo špatného nastavení program skončí.

- **Čekání na spojení / Přijaté spojení / Vytvoření nového vlákna** – V těchto stavech se program ocitne po přijetí příchozího spojení. Pomocí *selectu* vybere soket z fronty a vytvoří nové vlákno, ve kterém je spojení obsluhováno. Potom se opět čeká na další připojení.

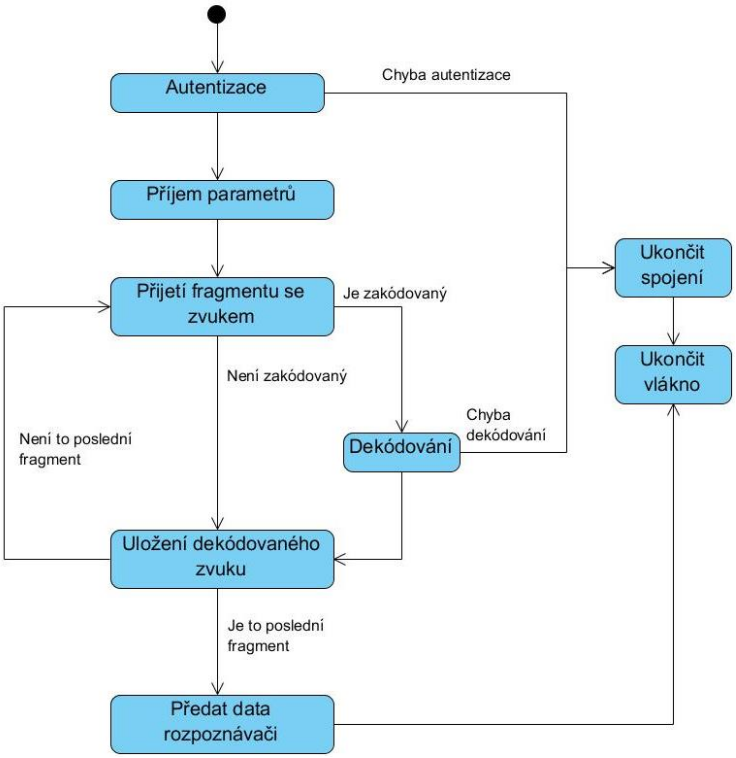


Obrázek 29: Spuštění serveru a přijímání příchozích spojení

Každé příchozí spojení je tedy přesunuto do vlastního vlákna, ve kterém je obsluhováno zvlášť. To je znázorněno na obrázku 30. Zpracování přijatého spojení je popsány těmito stavy:

- **Autentizace** – Jako první proběhne autentizace, která slouží k ověření, zda má připojený uživatel právo využívat tuto službu.
- **Příjem parametrů** – Jedná se o nepovinný krok. Všechny parametry už jsou defaultně nastaveny. Tady si můžeme nastavit například použitý zvukový kodek (PCM, speex, experimentálně opus), jazyk rozpoznávače řeči (CZ nebo EN – defaultně). Tento krok je povinný pouze při rozpoznávání fonémů, kde přijímá AMI fonémy.
- **Přijetí fragmentu se zvukem / Dekódování / Uložení** – Aplikace odesílá části zvuku už během nahrávání, server je přijímá a je-li potřeba, tak i dekóduje. Zatím si přijaté pakety střádá a pošle je rozpoznávači jako celek. V budoucnu je ale naplánovaná verze, která nic ukládat nebude, ale bude hned rozpoznávat. Aplikace je na tuto možnost připravena, bohužel zadaný modul nebyl v plně funkční verzi prozatím k dispozici.
- **Předat data rozpoznávači** – V tomto kroku se předají uložená data jako celek rozpoznávači. Ten je buď začne ihned zpracovávat, nebo si je uloží do fronty, než bude mít čas na jejich

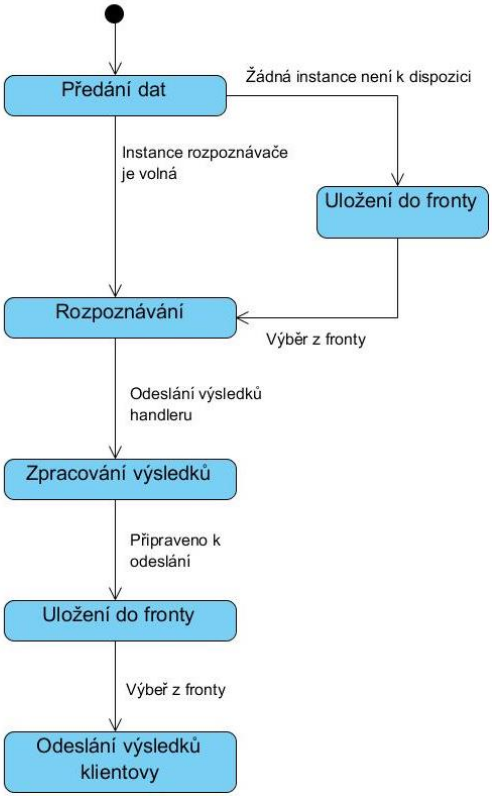
vyhodnocení (záleží na počtu volných instancí). Tento krok je asynchronní, proto je ukončeno vlákno daného připojení (ale není ukončeno spojení).



Obrázek 30: Zpracování přijatého spojení

Poslední částí je už pouze samotné rozpoznávání a odeslání výsledků. To je zobrazeno na obrázku 31 a je tvořeno těmito stavy:

- **Předání dat / Uložení do fronty / Rozpoznávání** – Rozpoznávač získá data od klienta a podle volných instancí provede ihned rozpoznání nebo si data uloží do fronty a počká, než se některá z instancí uvolní.
- **Zpracování výsledků** – Rozpoznávač odešle výsledky a v aplikaci je pomocí třídy *SOfflineSpeechRecCallbackI* zachytíme. Navíc zde provádím kontrolu zdvojených výsledků a prázdných polí. Někdy totiž rozpoznávač vrátí více stejných hodnot nebo dokonce prázdný řetězec.
- **Uložení do fronty –** V aplikaci je ještě jedno vlákno, které je uspané a čeká, než budou k dispozici výsledky rozpoznávání. V předchozím kroku uložíme data pro odeslání do fronty a probudíme toto vlákno. Dokud není fronta prázdná, tak odesílá data jednotlivým uživatelům. Pokud ji vyprázdní, tak se vlákno samo uspí a čeká na signál k probuzení.

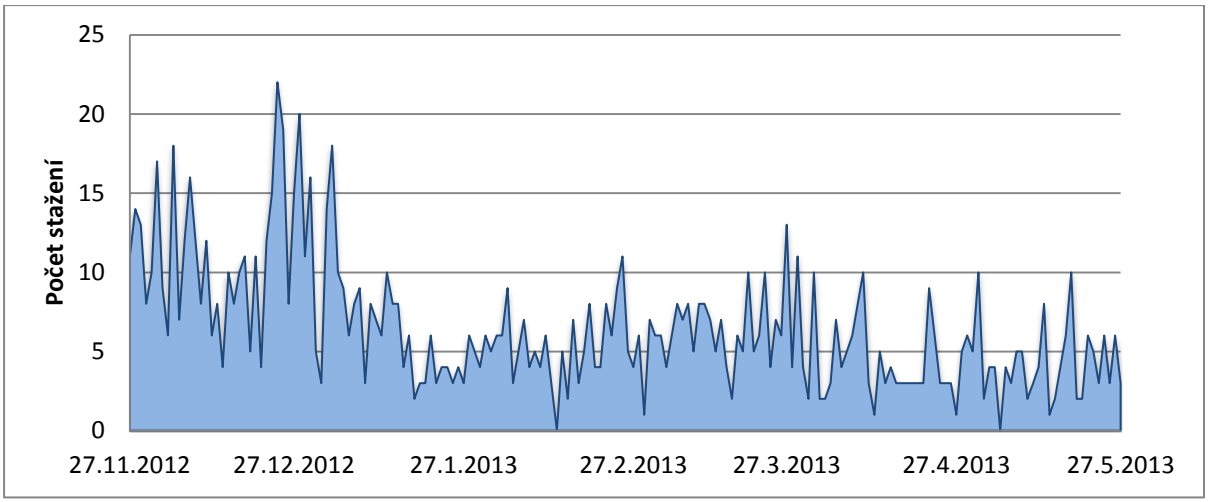


Obrázek 31: Použití rozpoznávače

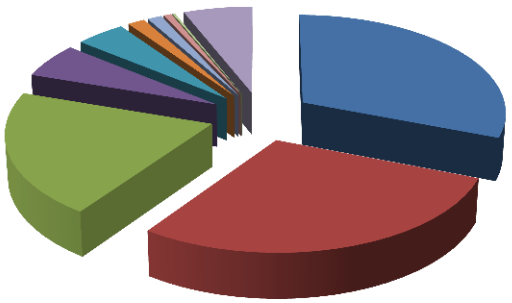
10 Testování

Prvním krokem implementace bylo vytvořit pouze funkční slovník (dle zadané specifikace) bez podpory hlasových technologií. Tato aplikace je už plně funkční a nachází se na Google Marketu. Google nabízí vývojářům i možnost zobrazit si statistiky prodeje či jen stahování jejich aplikací. Máme možnost si prohlédnout údaje o denním stahovaní, verzích systému Android, na kterých je aplikace nainstalována, nebo dokonce lze najít i konkrétní typ zařízení (např.: Samsung Galaxy S3).

Při pádu aplikace na straně uživatele zašle zařízení (je-li připojeno k internetu) zprávu s chybou, která pád způsobila. Tu si potom může prostřednictvím jejich sytému najít a opravit. Graf 2 ukazuje počty denních stažení free verze aplikace. Mimo to jsou k dispozici i placené verze s většími slovníky a více funkcemi.



Graf 2: Denní instalace aplikace



Android 2.3.3 - 2.3.7	807	30.86%
Android 4.0.3 - 4.0.4	755	28.87%
Android 4.1	534	20.42%
Android 4.2	149	5.70%
Android 2.2	115	4.40%
Android 3.2	40	1.53%
Android 2.1	33	1.26%
Android 3.1	18	0.69%
Android 1.6	6	0.23%
Ostatní	158	6.04%

Graf 3: Verze systému Android, na kterých je aplikace nainstalována

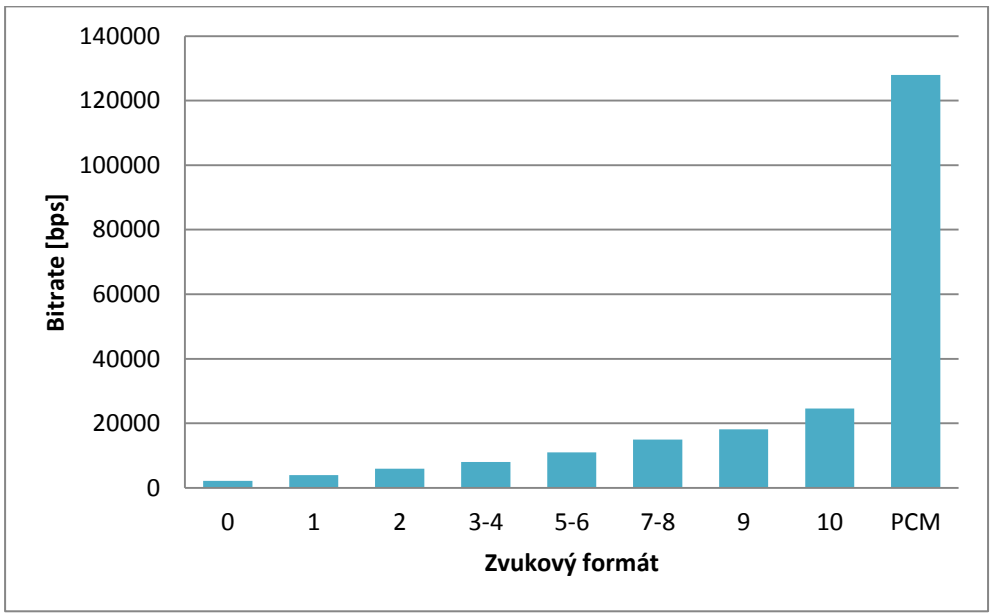
V návrhu aplikace byl požadavek, aby verze použitého API byla co nejnižší a neutrpěla tím funkčnost aplikace. Z grafu 3 je vidět, že největší zastoupení mají uživatelé se systémem Androidu 2.3.x. Tato verze systému využívá API verzi 8. Aplikace pracuje s verzí 4 (Android 1.6), proto bude pracovat i na starších systémech Android.

10.1 Kvalita použitého kodeku

V implementaci jsem ukazoval, proč je neefektivní posílat data bez komprese. Za prvé čistý PCM formát zabírá mnoho místa, proto se bude posílat více rámců. A další nevýhodou je zdejší připojení k internetu. V České republice zatím není mobilní internet běžnou věcí. Bývá hodně omezen (FUP) nebo se platí za přenesená data, což je u nás oproti jiným zemím zatím ještě docela drahé. Proto jsou posílané zvuky kódovány do speexu a zabírají méně místa.

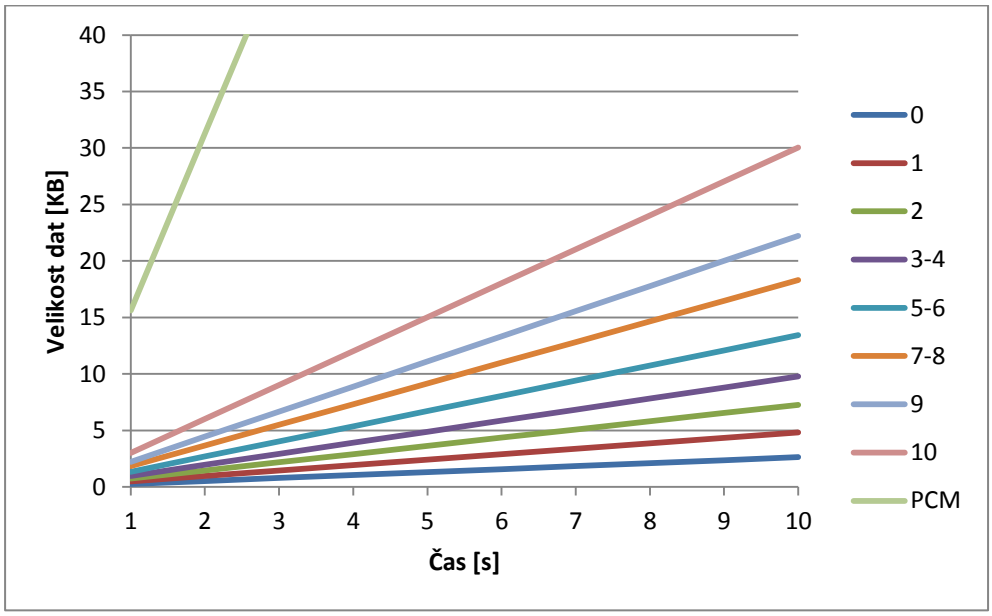
Kvalita	SPEEX (bps)	PCM (bps)	Kompresní poměr
0	2150	128000	59:1
1	3950		32:1
2	5950		21:1
3-4	8000		16:1
5-6	11000		11:1
7-8	15000		8:1
9	18200		7:1
10	24600		5:1

Tabulka 3: Komprese zvuku při použití speexu



Graf 4: Grafické zobrazení komprese zvuku při použití speexu

Tabulka 3 a graf 4 ukazují, jaké komprese je možné dosáhnou, při použití různých módů kódování do speexu. Vidíme, že při kvalitě 0 ušetříme téměř až 60 násobek původní velikosti. Čím vyšší číslo kvality, tím se kompresní poměr zhoršuje, ale musíme dbát i na kvalitu posílaného zvuku. V grafu 5 je znázorněno nahrávání 10 vteřin zvuku a jeho velikost při různém stupni kódování. Zatím to kódování nemá velké využití (při občasném využívání rozpoznávače), protože se na server přenáší pouze slova (max. 10 sekund), ale do budoucna se to vyplatí (rozpoznávání frází a vět) a aplikace je na tuto možnost už připravena.



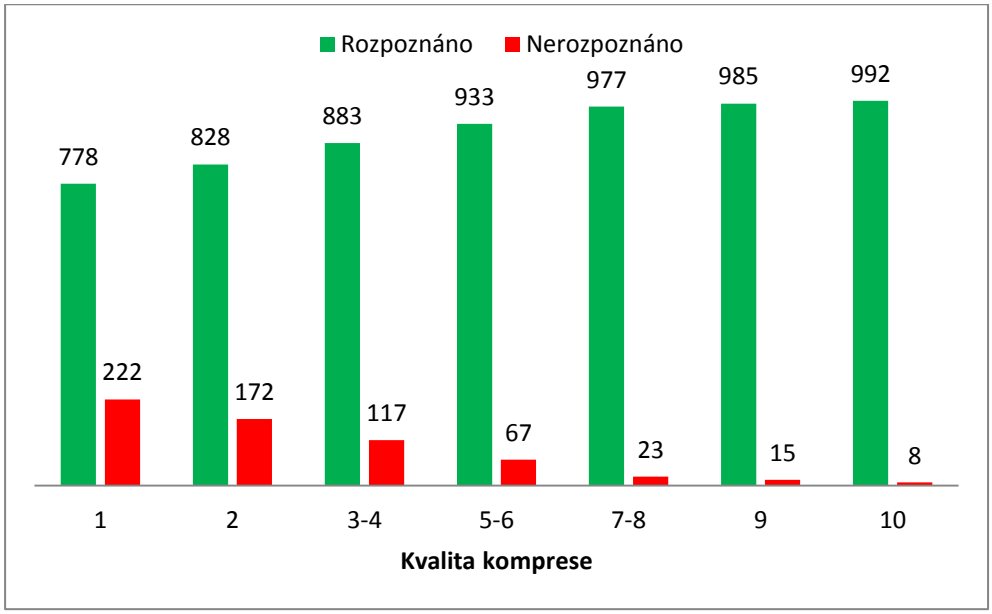
Graf 5: Velikost nahrávaných dat při použití komprese

Celková úspěšnost rozpoznávání slov byla kolem 80 až 90 procent. Měl jsem k dispozici množinu zvuků namluvených rodilým mluvčím, které jsem potom posílal na sever ve formátu PCM. Výsledky, které vrátily správné slovo na prvním místě, jsem si ukládal a potom jsem z nich náhodně vybral 1000 vzorků na další testování. Bylo potřeba zjistit jaká je nejvhodnější kvalita kódování zvuku. Snažil jsem se získat co nejlepší kompresní poměr a neztratit moc z funkcionality.

Kvalita	**Pozice ve výsledku**									**Rozpoznáno**	**Nerozpoznáno**	**Úspěšnost**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**			
1	563	118	53	12	11	8	7	2	4	778	222	77,8%
2	607	112	50	22	20	5	6	5	1	828	172	82,8%
3-4	660	132	41	17	14	8	6	3	2	883	117	88,3%
5-6	766	98	45	9	6		4	2	3	933	67	93,3%
7-8	859	83	19	7	4	1	1	1	2	977	23	97,7%
9	876	78	21	4	2	1	3			985	15	98,5%
10	947	38	6				1			992	8	99,2%

Tabulka 4: Úspěšnost rozpoznávače při použití komprese zvuku

Těchto 1000 vzorků jsem postupně překódoval do speexu ve všech kvalitách. Tím jsem dostal 7 sad po 1000 vzorcích a v různé kvalitě. Nakonec jsem je odeslal na server a zpracoval výsledky rozpoznávače. Cílem bylo, zjistit jak moc se zhorší rozpoznávání, když snížíme kvalitu zvuku. Rozpoznávač vracel devět možností rozpoznaného slova. V tabulce 4 jsou zobrazeny výsledky tohoto testu. Nakonec jsem zvolil kódování v kvalitě 7-8, protože má celkem velkou úspěšnost a většinu správných slov vrací už na prvních třech pozicích. Graf 6 pouze ukazuje celkovou úspěšnost všech použitých módů.



Graf 6: Úspěšnost rozpoznávače při použití komprese zvuku

11 Závěr

Cílem této práce bylo seznámit se s mobilní platformou Android, a dostupnými hlasovými technologiemi pro mobilní zařízení. Navrhnout a implementovat aplikaci, která bude sloužit jako slovník a bude obohacena funkcemi pro zpracování řeči. V aplikaci se využívají funkce pro převod řeči na text a pro vyhledání hranic fonémů ve slovech. Potřebné třídy pro implementaci těchto dvou funkcí jsou obsaženy v knihovně BSAPI.

Práce se v prvních kapitolách zabývá uvedením do problematiky zpracování zvukového signálu, strukturou rozpoznávačů řeči a jejich možným využitím v dnešní době. Následuje seznámení s platformou Android, návrhem aplikace, popisem její implementace a nakonec testováním.

Struktura celé aplikace byla navržena tak, aby bylo dosaženo co možná nejlepších výsledků. Na mobilním zařízení pracuje aplikace jako slovník a vlastní rozpoznávání probíhá na straně serveru. Dnešní telefony mají sice velký výkon, ale je potřeba si do paměti nahrát zvukové modely a další potřebná data k rozpoznávání. Jelikož rozpoznávač běží na serveru, tak je možné ho využít i jinými aplikacemi. Firma, pro kterou tento slovník vytvářím, začala s distribucí slovníků i na jiné mobilní platformy (Window Mobile, iOS), které tento rozpoznávač taky využívají. Prozatím se jedná pouze o testování, ale do budoucna je to plánováno začlenit jako stálý prvek těchto slovníků.

Mobilní aplikace je navíc implementována pouze jako takový skelet, do kterého se musí ještě nainstalovat data. Proto je potřeba ještě jeden server, který se stará o vytváření podepsaných balíčků s daty. Díky tomuto systému stačí aplikaci pouze data, podle kterých se z ní stane konkrétní slovník např. anglicko-český nebo německo-český. Tento systém nám navíc umožňuje k již nainstalovanému slovníku zakoupit a stáhnout další nové slovníky.

Komunikace aplikace se serverem pro rozpoznávání řeči probíhá pomocí spolehlivého protokolu TCP. Dalším budoucím krokem bude nahradit tento protokol nějakým rychlejším protokolem, který je přizpůsobený mobilním aplikacím. Posílaný zvuk je navíc kódován do speexu, čímž aplikace snižuje velikost posílaných dat. Aplikace je navíc navržena tak, že rozpoznávaní bude probíhat v „real-timu“. To znamená, že se nahrávaný zvuk odesílá na server po částech už během jeho záznamu. Na serveru jsou zatím data dekódována a uložena. Po skončení vysílání je teprve proveden převod a vrácen výsledek. Funkce, která by měla dělat okamžitý převod (data se na serveru neukládají, ale jsou hned poslány rozpoznávači), nebyla během implementace plně funkční, proto je to řešeno tímto způsobem.

Server zvládne převádět na text zatím pouze český a anglický jazyk. Rozpoznávač fonémů pracuje pouze s anglickým jazykem. Úspěšnost rozpoznávání byla nad 80 procenty a samotné rozpoznávání trvá asi tak dlouho, jako je délka nahrané zprávy (čeština je mnohem rychlejší). Při testování kvality přijatého zvuku jsem zjistil, že je tam dost šumu a zvuku z okolí. To by bylo možné ještě odstranit a zvýšit tak úspěšnost rozpoznávání.

Aplikace neslouží pouze k testování a experimentování, ale je navržena pro běžné použití. Použitelná základní verze (bez rozpoznávačů) je už k dispozici na Google Marketu. Nabízí vše, co bylo v této práci popsáno, kromě funkce rozpoznávání řeči, která do budoucna přibyde.

Literatura

[1] PSUTKA, Josef, Luděk MÜLLER, Jindřich MATOUŠEK a Vlasta RADOVÁ. *Mluvíme s počítačem česky*. 1. vyd. Praha: Academica, 2006, 752 s. ISBN 80-200-1309-1.

[2] Fonetika a Fonologie. *Akustická a auditivní fonetika* [online]. 2008 [cit. 2013-01-06]. Dostupné z: http://is.muni.cz/do/1499/el/estud/ff/js08/fonetika/ucebnice/ch06.html

[3] ČERNOCKÝ, Jan. *Zpracování řečových signálů - Studijní opora*. Brno: FIT VUT, 2006. Dostupné z: http://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf

[4] KREJSA, Jiří. *Digitalizace*. Brno: UMT FME VUT, 2010. Dostupné z: http://www.umt.fme.vutbr.cz/~ruja/vyuka/ZZS/DSP03.pdf

[5] ZAJÍC, Zbyněk. *Automatická adaptace akustického modelu*. Plzeň, 2008. Odborná práce ke státní doktorské zkoušce. Západočeská univerzita v Plzni. Dostupné z: http://www.kky.zcu.cz/cs/publications/ZbynekZajic_2008_Automatickaadaptace.

[6] *Crescendo Systems* [online]. 2007 [cit. 2013-01-06]. Dostupné z: http://www.crescendo.com/en/speech_recognition.php

[7] *SpeechTech MegaWord* [online]. 2013 [cit. 2013-01-06]. Dostupné z: http://www.speechtech.cz/cs/produkty/megaword.html

[8] *Novasoft* [online]. 2006 [cit. 2013-01-06]. Dostupné z: http://www.ccnovasoft.cz/cz/novavoice/

[9] *IOS: Siri* [online]. 2013 [cit. 2013-01-06]. Dostupné z: http://www.apple.com/ios/siri/

[10] *Nuance* [online]. 2002 [cit. 2013-01-06]. Dostupné z: http://www.nuance.com/

[11] Google Now. *phoneArena.com* [online]. 2012 [cit. 2013-01-06]. Dostupné z: http://www.phonearena.com/news/How-Google-Now-was-built-and-a-bit-on-Siri-too_id32033

[12] *Microsoft TellMe* [online]. 2013 [cit. 2013-01-06]. Dostupné z: http://www.microsoft.com/en-us/Tellme/default.aspx

[13] *S Voice* [online]. 2012 [cit. 2013-01-06]. Dostupné z: http://www.samsung.com/global/galaxys3/svoice.html

[14] *Svět aplikací: Podíl systému Android na americkém trhu dosáhl 52 procent* [online]. 2013 [cit. 2013-05-25]. Dostupné z: http://svetaplikaci.tyden.cz/podil-systemu-android-na-americkem-trhu-dosahl-52-procent/

[15] *Android developer: Dashboards* [online]. 2013 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[16] *Android developer: System Architecture* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/images/system-architecture.jpg

[17] LEE, Wei-Meng a Lauren DARCEY. *Beginning Android application development: průvodce programováním mobilních aplikací*. Vyd. 1. Indianapolis, IN: Wiley Pub., c2011, xx, 428 p. Wrox beginning guides. ISBN 978-111-8087-800.

[18] MEIER, Reto a Lauren DARCEY. *Professional Android application development*. 2nd ed. Indianapolis, IN: Wiley, 2009, xxviii, 761 p. ISBN 978-047-0344-712.

[19] MURPHY, Mark L a Lauren DARCEY. *Android 2: průvodce programováním mobilních aplikací*. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[20] *Android developer: Application Fundamentals* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/components/fundamentals.html

[21] *Android developer: Activities* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/components/activities.html

[22] *Android developer: Back Stack* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/images/fundamentals/diagram_backstack.png

[23] *Android developer: Activity Lifecycle* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/images/activity_lifecycle.png

[24] *Android developer: Services* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/components/services.html

[25] *Android developer: Service Lifecycle* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/images/service_lifecycle.png

[26] *Android developer: Content Providers* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/topics/providers/content-providers.html

[27] *Android developer: Exploring the SDK* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/sdk/exploring.html

[28] *Android developer: Tools Help* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/tools/help/index.html

[29] *Android developer: Android Developer Tools* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/tools/help/adt.html

[30] *Android developer: Android NDK* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/tools/sdk/ndk/index.html

[31] CONDER, Shane a Lauren DARCEY. *Android wireless application development*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, c2011, xxviii, 761 p. ISBN 03-217-4301-6.

[32] WAGGENER, William N a Lauren DARCEY. *Pulse code modulation techniques: with applications in communications and data recording*. Vyd. 1. New York: Van Nostrand Reinhold, c1995, xiii, 368 p. Wrox beginning guides. ISBN 04-420-1436-8.

[33] *Speex: Codec description* [online]. 2007 [cit. 2013-05-25]. Dostupné z: http://www.speex.org/docs/manual/speex-manual/node4.html

[34] *Speex: Introduction to Celp Coding* [online]. 2007 [cit. 2013-05-25]. Dostupné z: http://www.speex.org/docs/manual/speex-manual/node9.html

[35] *Speex: Coding* [online]. 2007 [cit. 2013-05-25]. Dostupné z: http://www.speex.org/docs/manual/speex-manual/img7.png

[36] *Speex: narrowband mode* [online]. 2007 [cit. 2013-05-25]. Dostupné z: http://www.speex.org/docs/manual/speex-manual/node10.html

[37] *Speex: wideband mode* [online]. 2007 [cit. 2013-05-25]. Dostupné z: http://www.speex.org/docs/manual/speex-manual/node11.html

[38] *Android developer: Android manifest* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[39] *Android developer: ProGuard* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/tools/help/proguard.html

[40] *Android developer: Screens Ranges* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/images/screens_support/screens-ranges.png

[41] *Android developer: Processes and Threads* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://developer.android.com/guide/components/processes-and-threads.html

[42] *Phonexia: Brno Speech Application Interface Documentation* [online]. 2012 [cit. 2013-05-25]. Dostupné z: http://www.phonexia.com/docs/bsapi/index.html

Seznam příloh

Příloha 1: CD s vlastní prací.

Zdrojové kódy nejsou součástí příloh, protože podléhají autorským právům společnosti Lingea s.r.o. a Fakulty Informačních technologií VUT v Brně. BSAPI je možné si na fakultě v případě zájmu vyžádat.

Obsah CD

- diplomova_prace.pdf – Tisknutelná verze práce
- diplomova_prace.docx – Zdrojová verze práce